THIRD EDITION

SUPERCHARGING MS DOS®

COVERS VERSION 5

The Microsoft® Guide to High-Performance Computing

Van Wolverton
Dan Gookin

Microsoft PRESS®

The Authorized Edition

SUPERCHARGING MS DOS®

COVERS VERSION 5

THIRD EDITION

Van Wolverton
Dan Gookin

Microsoft PRESS®

Framingham State College
Framingham, Massachusetts

PUBLISHED BY
Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Copyright © 1991 by Van Wolverton

All rights reserved. No part of the contents of this book
may be reproduced or transmitted in any form or by any means
without the written permission of the publisher.

Library of Congress Cataloging-in-Publication Data
Wolverton, Van, 1939–
Supercharging MS-DOS / Van Wolverton, Dan Gookin -- 3rd ed.
p. cm.
Includes index.
ISBN 1-55615-371-6 : $24.95 ($32.95 Canada)
1. Operating systems (Computers) 2. MS-DOS (Computer file)
I. Gookin, Dan. II. Title.
QA76.76.O63W655 1991
005.4'46--dc20 91-22342
CIP

Printed and bound in the United States of America.

1 2 3 4 5 6 7 8 9 AGAG 6 5 4 3 2 1

Distributed to the book trade in Canada by Macmillan of Canada,
a division of Canada Publishing Corporation.

Distributed to the book trade outside the United States and Canada by Penguin Books Ltd.

Penguin Books Ltd., Harmondsworth, Middlesex, England
Penguin Books Australia Ltd., Ringwood, Victoria, Australia
Penguin Books N.Z. Ltd., 182–190 Wairau Road, Auckland 10, New Zealand

British Cataloging-in-Publication Data available.

Epson® is a registered trademark of Epson America, Incorporated.
AT®, IBM®, and PS/2® are registered trademarks and PC/XT™ is a trademark of
International Business Machines Corporation.
Microsoft® and MS-DOS® are registered trademarks of Microsoft Corporation.
ProKey® is a registered trademark of RoseSoft.
SideKick® is a registered trademark of Borland International, Incorporated.
Teletype® is a registered trademark of Teletype Corporation.
WordStar® is a registered trademark of MicroPro International Corporation.

Acquisitions Editor: Michael Halvorson
Manuscript Editor: Lisa Kleinman
Technical Editor: Jerry Joyce

QA
7676
063
W655
1991

For Edward, Katherine, and Andrew,
Inheritors of the Faith, Keepers of the Flame,
Guardians of the Trust:
You'll never know just how much
I love you.

CONTENTS

Introduction ix

SECTION I Going Beyond the Basics

CHAPTER 1
Let's Get Technical: Of Bytes, ASCII, and Hexadecimal 3

CHAPTER 2
Using the Extended Character Set 11

CHAPTER 3
Taking Control of Your Screen and Keyboard with ANSI.SYS 19

CHAPTER 4
Learning Your Printer's Language 41

CHAPTER 5
Debug: A Special Sort of Editor 57

CHAPTER 6
Advanced Batch-File Techniques 83

CHAPTER 7
Designing an Interactive Menu System 111

SECTION II Customizing Your System

CHAPTER 8
Controlling the Environment and CONFIG.SYS 143

CHAPTER 9
Getting the Most from Memory 159

CHAPTER 10
Setting Up and Using a RAM Disk 185

CHAPTER 11
Display It Your Way 203

CHAPTER 12
Print It Your Way 223

CHAPTER 13
Increasing Your Keyboard's IQ 259

CHAPTER 14
Playing It Smart with Files and Disks 289

CHAPTER 15
Putting It All Together: Your Own Menu System 329

CHAPTER 16
A Survival Guide for the Nineties 353

CHAPTER 17
The Care and Feeding of Your Computer 389

SECTION III Quick Reference

APPENDIX A
A Quick Reference to ANSI.SYS Commands 401

APPENDIX B
Epson-Compatible and HP-Compatible Printer Commands 409

APPENDIX C
The ASCII and IBM Extended Character Sets 415

APPENDIX D
ANSI.SYS Key and Extended Key Codes 421

APPENDIX E
Converting Hexadecimal Numbers to Decimal 427

APPENDIX F
A Sample Menu System 431

Glossary 441
Index 449

INTRODUCTION

If you have used your computer for a few months, you know there's more to DOS than C>. But do you know how *much* more? This book shows you how, using nothing more than DOS, you can:

- Design your own screens, using bold, reverse video, and color, to display menus, help information, or instructions.
- Create, examine, or change *any* file, even if it contains a program.
- Access extended, expanded, and upper memory, and take advantage of all the other advanced features of DOS 5.
- Set up a customized menu system that lets you run any of your application programs by pressing only one or two keys.

You'll see how to do all that, and more. But this book doesn't describe routine uses of DOS; it assumes that you have used DOS long enough to be familiar with its commands, have written some batch files, and want to know more. *Running MS-DOS,* also published by Microsoft Press, introduces MS-DOS and shows you how to use its commands. If you're new to computers, you should start with that book; this book picks up where *Running MS-DOS* leaves off.

Like *Running MS-DOS,* however, this book doesn't assume that you want to become a programmer. It assumes that your goal is to take advantage of the advanced capabilities of DOS to use the computer more productively. In keeping with this goal, the book is written in plain English and uses real-life examples.

About the Batch Files and Programs

The book describes many batch files and programs. Some are short, most contain 10 to 20 lines, and a few are longer. It takes some time to type and test them, but you should find several that will make your life easier. Except for one program in Chapter 13 that works only with the IBM PC/AT and IBM PS/2 computers—and computers compatible with those models—the batch files and programs work with any IBM or IBM-compatible computer.

Although you can buy special-purpose utility programs that provide many of the capabilities covered by the batch files and programs in this book, you may find that you don't need them. You can easily tailor the batch files

shown here to your own needs or preferences. Not only can you make your computer a more efficient tool, but you also get the satisfaction of knowing that you did it yourself and that, if the need arises, you can change it again.

And it's all—well, *almost* all—DOS.

What's in the Book and Where

Section I includes Chapters 1 through 7. These chapters describe some general techniques for advanced DOS uses. Chapter 1 defines *byte, hexadecimal,* and *ASCII.* Chapter 2 shows you how to use the IBM extended character set, and Chapter 3 shows you how to use the ANSI.SYS device driver to control the display and keyboard. Chapter 4 shows you how to use Epson– and Hewlett Packard–compatible printer commands. Chapter 5 shows you how to use the Debug program to create small assembly language programs. Chapters 6 and 7 show you some advanced batch-file techniques and then apply these techniques to creating a menu system.

Section II includes Chapters 8 through 17. These chapters apply the techniques described in Section I to specific parts of your computer system: They show you how to create and use a number of batch files and programs. Chapter 8 shows you how to control the environment and configuration; Chapter 9 tells you about memory and how to put it to work under DOS 5; and Chapter 10 shows you how to set up and use a RAM disk and disk cache. Chapters 11, 12, and 13 cover the display, printer, and keyboard, and Chapter 14 covers files and disks. Chapter 15, drawing on techniques described in most of the preceding chapters, shows you how to create a complete menu system tailored to the application programs you use. Chapter 16 explains how to use a DOS computer—based on 10-year-old technology—with today's modern hardware, plus a few other survival tips. Chapter 17 concludes Section II by suggesting some ways to make your environment safer and more comfortable for both your computer and yourself.

Appendixes A through F provide additional reference material. Appendixes A and B are quick references to ANSI.SYS commands and the more common printer commands. Appendixes C and D are quick references to the ASCII and IBM extended character sets and the key codes used with ANSI.SYS commands. Appendix E describes converting hexadecimal numbers to decimal numbers, and Appendix F shows the contents of all the files needed for the menu system described in Chapter 14.

How You Can Use this Book

Read Section I first. It provides a foundation of techniques used in the batch files and programs described in Section II. You don't need to read the chapters in Section II in order. If you're especially interested in the printer, for example, read Chapter 12 first. Where knowledge of something in a previous chapter is required, a cross-reference is provided.

The menu system described in Chapter 15 draws from almost every other chapter; more than anything else in this book, it demonstrates how much you can do with DOS. If you tailor the menu system to the programs you use, you can run any application in any directory by pressing only one or two keys.

The menu system requires more effort than any other batch file in the book, but once you set it up, you may never again have to type a command to change the directory or start an application program. Because Chapter 15 draws so heavily on the other chapters in the book, however, it is probably better to wait until you have gone through most of the book before you tackle the menu system.

Differences Between DOS Versions

DOS has been revised several times. A change in the number *following* the decimal point—3.2 to 3.3, for example—marks a minor change that leaves DOS substantially the same as the previous version. A change in the number *preceding* the decimal point, however, marks a major change. Version 2.0, for example, had almost three times as many commands as version 1.1.

Unless otherwise noted, the commands and responses in this book are based on version 5 of DOS, but the book can be used with all versions of DOS since 2.0 and all models of IBM and IBM-compatible computers.

The following list shows the significant additions made by each of the minor versions of version 3, and the additions made by versions 4 and 5:

Version 3.0
File sharing for network and multitasking systems
Share command
Attrib command
VDISK.SYS and RAMDRIVE.SYS device drivers
<drive>:<path> allowed before program name
<path> allowed before file name in IF EXIST command

Version 3.1
Support for the IBM PC Network
Join command
Subst command

Version 3.2
Xcopy command
Replace command
Append command
DRIVER.SYS, a device driver for nonstandard disk drives
Support for 3.5-inch disk drives

Version 3.3
Fastopen command
Nlsfunc command
Chcp command
Call command
10 other commands changed

Version 4
Mem command
DOSSHELL.BAT, which runs the DOS Shell program
Select command changed to an installation program
XMAEM.SYS and XMA2EMS.SYS, expanded memory device drivers
Hard-drive partitions larger than 32 MB
18 other commands and 4 other device drivers changed

Version 5.0
Addition of /? switch for help on DOS commands
QBasic programming environment
Edit, a full-screen, mouse-supported text editor
Doskey command-line editing and macro utility
Support for upper memory and extended and expanded RAM
Improved DOS Shell program
Sorting and attribute options added to the Dir command
Attrib command modified to hide files
Safe formatting of disks
Undelete utility
Plus upgrades and improvements to many DOS commands

What to Type and When

This book uses the following conventions to show what you type and how DOS responds:

- Hands-on examples are shown in different type, on separate lines, just as you would see them on your display. Characters you type are printed in lowercase italic. Here is an example that shows this convention:

```
C>path
PATH=C:\;C:\DOS
```

- When characters you type appear in the text, the interaction between you and the computer is printed in italics to distinguish it from the surrounding text. For example: "The commands are *prompt $e[1;36m* for high-intensity cyan and *prompt $e[m* to return the display attributes to normal."

- The names of keys in the examples are shown in italics in angle brackets (<>) to distinguish them from characters that you type: <*Ctrl-Break*>, for example, means hold down the Ctrl key and press the Break key.

- The book shows the system prompt as *C>*, as in the first example. DOS displays something different if the current drive is not drive C or if you have defined your own system prompt. Whenever you see *C>* in the book, assume that it refers to your system prompt.

- Many commands include parameters that allow you to specify a particular disk drive, device, or other option. Parameters are shown in italics when they represent a variable entry, such as the name of a file. When parameters must be entered exactly, they are shown in roman type. For example, here are the parameters of the Copy command:

  copy *file1 file2*

  The word *copy* is required and must be entered as shown. *file1* represents the name of the source file, and *file2* represents the name of the target file.

- In this book, the Escape character is represented by the symbol [E]. Refer to the section "Entering the Escape character in a file" in Chapter 3 to learn how to enter the Escape character with your word processor. Use the string *$e* to represent the Escape character in the Prompt command.

Enough preparation; it's time to dig into DOS. Turn on your computer and turn to Chapter 1.

SPECIAL OFFER

Companion Disk for SUPERCHARGING MS-DOS,® 3rd ed.

Microsoft Press has created a timesaving Companion Disk to save you the time of typing in the various files and programs in the book—and the additional time of finding and correcting those inevitable typing errors. Available in both 5 1/4-inch and 3 1/2-inch format, the disk contains all the batch files, script files and programs in SUPERCHARGING MS-DOS, 3rd ed., and was designed to be used with the book. Get started right away, order your copy today!

Domestic Ordering Information:
To order, use the special reply card in the back of the book. If the card has already been used, please send $19.95, plus sales tax in the following states if applicable: AZ, CA, CO, CT, DC, FL, GA, HI, ID, IL, IN, IA, KS, KY, ME, MD, MA, MI, MN, MO, NE, NV, NJ, NM, NY, NC, OH, OK, PA, RI, SC, TN, TX, VA, WA, WV, WI. Microsoft reserves the right to correct tax rates and/or collect the sales tax assessed by additional states as required by law, without notice. Please add $2.50 per disk set for domestic postage and handling charges. Mail your order to: **Microsoft Press, Attn: Companion Disk Offer, 21919 20th Ave SE, Box 3011, Bothell, WA 98041-3011**. Specify 5 1/4-inch or 3 1/2-inch format. Payment must be in U.S. funds. You may pay by check or money order (payable to Microsoft Press) or by American Express, VISA, or MasterCard; please include credit card number, expiration date, and cardholder signature. Allow 2–3 weeks for delivery.

Foreign Ordering Information (except within the U.K. and Canada, see below):
Follow procedures for domestic ordering. Add $15.00 per disk set for foreign postage and handling.

U.K. Ordering Information:
Send your order in writing along with £17.95 (includes VAT) to: Microsoft Press, 27 Wrights Lane, London W8 5TZ. You may pay by check or money order (payable to Microsoft Press) or by American Express, VISA, MasterCard, or Diners Club; please include credit card number, expiration date, and cardholder signature. Specify 5 1/4-inch or 3 1/2-inch format.

Canadian Ordering Information:
Send your order in writing along with $26.95 (includes GST) to: Macmillan Canada, Attn: Microsoft Press Department, 164 Commander Blvd., Agincourt, Ontario, Canada M1S 3C7. You may pay by check or money order (payable to Microsoft Press) or by VISA or MasterCard; please include credit card number, expiration date, and cardholder signature. Specify 5 1/4-inch or 3 1/2-inch format.

Microsoft Press Companion Disk Guarantee:
If a disk is defective, a replacement disk will be sent. Please send the defective disk along with your packing slip (or copy) to: Microsoft Press, Consumer Sales, One Microsoft Way, Redmond, WA 98052-6399.

Send your questions or comments about the files on the disk to
Supercharging MS-DOS, 3rd ed., Microsoft Press,
One Microsoft Way, Redmond, WA 98052-6399

The Companion Disk for
SUPERCHARGING MS-DOS, 3rd ed.
is available only from Microsoft Press.

SECTION

I

Going Beyond the Basics

CHAPTER

1

Let's Get Technical: Of Bytes, ASCII, and Hexadecimal

You can use most application programs and DOS commands without knowing about bytes, ASCII codes, or hexadecimal numbers. So why should you care about them? Because knowing something about these subjects will help you become more adept at using DOS. And this book describes some DOS commands and programs that require at least a familiarity with these concepts. Even if you only use your computer to run the same few application programs, the techniques that are covered in this book can make your system easier to use and make you more productive.

What Is a Byte?

You know that computer memory and disk capacity are measured in bytes, and that a byte can store a character, but just what is a byte?

Computer memory is made up of thousands of electronic switches (usually transistors). Because these switches can be either on or off, the numbers 1 and 0 are used to represent them. The base-2 number system—whose digits are 1 and 0—is called binary, so each switch represents one binary digit, or bit. A byte is a group of eight of these switches, or bits.

We're talking about small switches here. A typical memory chip has 262,144 of them, giving it a capacity of 32,768 bytes. A computer with 640K bytes of memory has 5,242,880 switches.

Because eight on-off switches can be arranged in 256 different combinations, a byte has 256 possible values, ranging from 0 (all switches off) through 255 (all switches on). DOS can quickly locate any byte in memory and either check the position of the eight switches (read the byte) or set the switches (write the byte). When you save a file on a disk, DOS reads the position of the switches in memory where the file is stored, then records the state of these switches on the disk. When you load a file from a disk, DOS reads the state of the switches from the disk, then sets the switches in memory to match.

Although the binary value of a byte can represent a character, a number, or even a program instruction, its meaning as a character is most important to this book. Almost all microcomputers throughout the world use the same code to translate byte values into the familiar letters, numbers, and symbols we use. It's known as the ASCII code.

The ASCII Code

The term ASCII (rhymes with *passkey*) frequently crops up in manuals, books, and articles about computers. Like many other computer terms,

ASCII is an acronym—it stands for American Standard Code for Information Interchange. The key word here is *code:* Like the Morse code used in radio and telegraphy, ASCII is an internationally accepted code for representing characters used in computers and telecommunications.

Because the value of a byte can range from 0 through 255, a byte can represent one of 256 possible characters. The ASCII standard defines the codes for the first 128 characters (codes 0 through 127). With few exceptions, any microcomputer in any country uses the same code to translate byte values into characters; the exceptions are usually characters that vary from country to country, such as the currency symbol ($ in the United States, £ in the United Kingdom, ¥ in Japan, and so forth).

The first 128 characters can be divided into two groups: Codes 0 through 31 are *control characters,* which define some action such as a carriage return or a tab. Codes 32 through 127 are the characters you normally see in text. The ASCII standard doesn't define the characters for codes 128 through 255. The characters displayed for those values differ on each computer, although all DOS-compatible systems show the same characters, called the *extended ASCII character set*.

Control characters

ASCII character codes 0 through 31 don't define any displayable text characters. Instead, those codes are used for special purposes and perform special functions.

Each control code is represented by a control character made up of a letter of the alphabet, A through Z, preceded by a special character. For example, code 1 is "Control-A," often abbreviated or displayed as ^A. The ^, or caret, stands for "control." Each control character has a name that refers to its function. Control-A is the "start of header" character, or "SOH." The 32 codes, their characters, and what they're called are shown in Figure 1-1.

ASCII Code	*Char.*	*Name*	
0	^@	NUL	Null
1	^A	SOH	Start of heading
2	^B	STX	Start of text
3	^C	ETX	End of text
4	^D	EOT	End of transmission
5	^E	ENQ	Enquiry
6	^F	ACK	Acknowledge
7	^G	BEL	Bell

Figure 1-1. *Control characters.* *(continued)*

Figure 1-1. *continued*

ASCII Code	*Char.*	*Name*	
8	^H	BS	Backspace
9	^I	HT	Horizontal tab
10	^J	LF	Linefeed
11	^K	VT	Vertical tab
12	^L	FF	Formfeed
13	^M	CR	Carriage return
14	^N	SO	Shift out
15	^O	SI	Shift in
16	^P	DLE	Data link escape
17	^Q	DC1	Device control 1
18	^R	DC2	Device control 2
19	^S	DC3	Device control 3
20	^T	DC4	Device control 4
21	^U	NAK	Negative acknowledge
22	^V	SYN	Synchronous idle
23	^W	ETB	End transmission block
24	^X	CAN	Cancel
25	^Y	EM	End of medium
26	^Z	SUB	Substitute
27	^[	ESC	Escape
28	^\	FS	File separator
29	^]	GS	Group separator
30	^^	RS	Record separator
31	^_	US	Unit separator

In the old days, ASCII control codes were used primarily by teletypes. Today only a few of the codes hold any real meaning for a computer.

What's interesting about these characters is how they are integrated into your keyboard and the functions they perform. Control-H, for example, is the ASCII code produced when you press the Backspace key. In fact, holding down the Control key and typing an H has the same effect as pressing the Backspace key. Other codes and keys worth noting are Control-I, which is the same as the Tab key; Control-[, the Escape key ("Esc" on most keyboards); Control-C, the Break key (Control-Pause or Control-Scroll Lock on most keyboards); and Control-M, the Enter key.

A few interesting control characters not associated with a particular key are Control-L, which when sent to your printer will eject a page, and Control-G, which causes the computer's (or printer's) speaker to beep. Type in the following at your DOS prompt:

```
C>echo <Ctrl-G>
```

(Press the Ctrl key and type a G; don't type in "<Ctrl-G>". Note that when you press Ctrl-G, you'll see "^G" displayed on your screen.)

Press Enter and you'll actually hear an ASCII character!

In addition to the control characters and their teletype functions, most IBM-compatible systems also have a group of 32 characters represented by codes 0 through 31. They aren't ASCII characters, and they're particular to DOS computers. The drawback to these characters is that you can see them on your screen only when using special programming methods. (This set of characters can be seen in Appendix C.)

Visible text characters

ASCII codes 32 through 127 represent the characters you see on your screen. This includes all letters of the alphabet (with separate codes for uppercase and lowercase letters) as well as punctuation symbols and numbers.

Take a look at Appendix C, Figure C-1, and note the ASCII codes from 32 through 127 and their corresponding characters. Keep your thumb there and turn back here when you're done.

Now take a second look at the ASCII table and note how ingeniously it's laid out. Uppercase letters are located at exact increments from their lowercase counterparts. Furthermore, the control characters share letters at similar increments. Control-A is code 1; add 64 to get code 65, or capital A; and add 32 to get lowercase a. (See how each column shows 32 characters? This is yet another aspect of computers and binary: 32 is 2 to the fifth power.)

This may appear only slightly interesting to you, but being able to recognize and use such patterns is an important part of understanding how your computer works.

The extended ASCII character set

There is no one standard for character codes 128 through 255. Each microcomputer, even though it may be fully ASCII-compatible with codes 0 through 127, will display different characters for those codes. On IBM-compatible systems, however, a standard known as the *extended ASCII character set* exists. These characters are the same on nearly all computers that use DOS.

The 128 characters in the IBM extended character set fall into four groups:

- *International characters* (codes 128 through 175), which are mostly accented characters, such as ç (code 135) and ñ (code 164).
- *Line-drawing and box-drawing characters* (codes 176 through 223), such as ╔ (code 201) and ╝ (code 188).
- *Greek letters* (codes 224 through 237), such as ∝ (code 224) and Φ (code 237).
- *Mathematical symbols* (codes 238 through 253), such as ± (code 241) and ✓ (code 251).

Books and manuals frequently include charts of the ASCII code, showing each byte value and its corresponding character. Your printer's manual may include more than one chart, because many printers let you choose from several character sets that offer different combinations of international characters, fonts, and graphics characters. This is also possible on your computer when you use DOS's ability to switch *code pages*. Each code page contains only slightly different characters, however.

Appendix C in this book shows the full IBM-compatible character set, including the ASCII standard characters and the IBM extended character set using the standard, American English–language code page. The code for each character is given in both decimal and hexadecimal.

Hexadecimal Numbers

Hexadecimal, or *hex,* is the name of the base-16 number system, just as decimal is the name of the base-10 number system we commonly use. We probably use decimal numbers because we have 10 fingers. Who uses a base-16 number system, and why?

Not surprisingly, the reason we use base-16 is related to the base-2 number system computers use. Because 2 and 16 are both powers of 2, hexadecimal is a shortcut way of working with binary numbers. The decimal number 178, for example, is 10110010 in binary, but only B2 in hexadecimal. That's a neat two-digit number versus a long eight-digit number. Programmers, engineers, and others who work with computers use hexadecimal to avoid all those ones and zeros.

There are 16 hexadecimal digits: 0 through 9 plus the letters A through F, representing the decimal values 10 through 15. A hexadecimal number can be F, which is 15 in decimal, or 10F (271 in decimal), or even FACE (64,206 in decimal).

Using a computer doesn't mean you have to learn hexadecimal, but it's difficult to completely avoid hexadecimal as you explore the more advanced areas of DOS. The Comp command, for example, reports the location and value of differing bytes in hexadecimal. The Debug program uses hexadecimal numbers exclusively, and printer commands that control graphics also require hexadecimal values.

For the most part, you needn't worry about hexadecimal arithmetic. The most you'll have to do in this book is recognize that a hexadecimal number is, indeed, a number, and perhaps type a few hexadecimal numbers. If you're interested, Appendix E describes hexadecimal numbers in more detail and includes a table that simplifies converting hexadecimal numbers to decimal numbers.

Chapter Summary

- The base-2 (binary) number system uses only two digits—1 and 0.
- A byte is made up of eight binary digits, or bits, and can hold a value from 0 through 255.
- The ASCII code is used to represent letters, numbers, and symbols.
- The ASCII code covers the first 128 characters (codes 0 through 127).
- IBM and IBM-compatible computers use the IBM extended character set (codes 128 through 255).
- Programmers use the base-16 (hexadecimal) number system because it's a shortcut way to use the binary system that computers use.
- Appendix C shows the ASCII code and the IBM extended character set.
- Appendix E describes hexadecimal in more detail and includes a hexadecimal-to-decimal conversion table.

CHAPTER

2

Using the Extended Character Set

Your computer displays many characters besides the letters, numbers, and punctuation marks on the keyboard. These additional characters belong to the IBM extended character set. This chapter shows you how you can enter and use these non-keyboard characters.

As Chapter 1 explains, character codes 0 through 31 represent control functions, codes 32 through 127 represent the standard ASCII text characters, and (on IBM and IBM-compatible computers) codes 128 through 255 represent international characters, line-drawing and box-drawing symbols, and mathematical symbols. Appendix C shows both the standard ASCII character set and the IBM extended character set.

Using the Alt Key to Enter a Non-Keyboard Character

You can't type the non-keyboard characters because the keyboard doesn't have keys for all 256 codes. But there is a way on most IBM-compatible computers to enter any of these characters directly from the keyboard: Hold down the key labeled *Alt*, type the character's code on the numeric keypad, and then release the Alt key.

Throughout this chapter—throughout the book, in fact—you'll use this technique to enter non-keyboard characters. Give it a try: Hold down the Alt key, type 171 on the numeric keypad (not on the top row of number keys), and then release the Alt key. DOS should respond by displaying the fraction ½; this is character 171.

This book uses the convention Ctrl-C to represent holding down the key labeled *Ctrl* and pressing C; it uses a similar convention to represent holding down the Alt key, typing a number on the keypad, and releasing the Alt key. The example in the previous paragraph would be shown as Alt-171.

Backspace to erase the fraction, and enter more non-keyboard characters:

```
C><Alt-232><Alt-233><Alt-234>
```

DOS should display the Greek letters Φ, θ, and Ω.

The Alt-key-plus-number technique works for any character, not just the non-keyboard characters. Press the Escape key to clear the command line, then type:

```
C><Alt-100><Alt-105><Alt-114>
```

100 is the code for d; typing Alt-100 is the same as typing *d*. 105 is the code for i, and 114 is the code for r. What you just typed looks like a DOS Directory command. Press Enter. It *is* a Directory command.

If you really had to, you could type everything using just the Alt key and the numeric keypad. Happily, you don't have to, but being able to type any character, whether or not it appears on the keyboard, will prove to be quite useful.

Using the International Characters

You can use the Alt-key technique to enter characters from another language. At the DOS prompt, type:

```
C><Alt-168>Por qu<Alt-130>?
```

DOS displays a question in Spanish:

```
C>¿Por qué?
```

Entering the Control Characters

You can use the Alt key to type control codes—characters 1 through 31 in the ASCII code—as well as characters. For example, the character 13 is a carriage return. Type another Directory command, but, instead of pressing the Enter key, type Alt-13:

```
C>dir<Alt-13>
```

The result is the same as if you had pressed the Enter key.

How Many Ways Are There to Type Something?

With DOS, you can perform some actions three different ways: You can use action keys (such as F6 or the Enter key), a Ctrl-key sequence (for example, Ctrl-C or Ctrl-S), or even an Alt-key sequence (such as Alt-26). So, if you want to cancel a command, you can press Ctrl-Break, Ctrl-C, or Alt-3. Figure 2-1 lists the DOS action keys and their associated Ctrl-key and Alt-key sequences.

Action Key	*Ctrl-Key*	*Alt-Key*
Ctrl-Break	Ctrl-C	Alt-3
Backspace	Ctrl-H	Alt-8
Tab	Ctrl-I	Alt-9
Enter	Ctrl-M	Alt-13
Ctrl-PrtSc	Ctrl-P	Alt-16
F6	Ctrl-Z	Alt-26

Figure 2-1. *Ctrl-key and Alt-key sequences.*

Because DOS echoes a control character as a caret (^) followed by the corresponding letter (regardless of how it is entered) and because most manuals and books use the Ctrl-key technique when entering these characters, this book, too, uses the Ctrl-key technique to represent typing character codes 1 through 26 (Ctrl-A through Ctrl-Z). The Alt-key technique is used to represent typing all other non-keyboard characters.

> ❑ **NOTE:** *Like DOS, Edlin lets you enter control characters with either the Ctrl key or Alt key, and always echoes with a caret followed by the appropriate letter. Some word processors and text editors, however, accept only one of these techniques. You'll have to experiment to see which method your program accepts.*

The first 26 ASCII characters are used for more than just DOS control characters. Most printers, for example, accept commands to control the type of printing or line spacing in a document. So that the printer can distinguish a command from text to be printed, the commands start with non-printing characters. You can control your printer by including these commands in the text.

For example, character 14 (Ctrl-N) tells an Epson-compatible dot-matrix printer to print one line in large type. Make sure your printer is on, and then type:

```
C>echo <Ctrl-N>This line is large. > prn
```

The line will be printed in large type.

Other printers, including the popular Hewlett-Packard line of laser printers (and those compatible with it), use different commands but retain control codes as a method of changing the type. For more information on printer commands, see Chapter 4, "Learning Your Printer's Language."

Searching for Control Characters with a Word Processor

In some instances, using the Alt-key technique may be the only way to achieve what you want. For example, it's sometimes difficult or impossible to search for or to replace control characters in a document using a word-processing program, even though you might want to do this. Suppose you want to search for a tab or a carriage return. Many word processors use those characters to select a search command option, or even to end the command, so you can't tell the program to search for them by pressing the Tab key or the Enter key.

Some word processors, however, such as Microsoft Word for Windows, let you search for a tab (character 9) by entering Alt-9 as the search character. When you enter the search character, the word processor might display the corresponding character shown in Appendix C, but it will search for the correct character.

Another popular technique for finding—or even inserting—control codes in a document is to use a special prefix character. In WordStar, for example, Ctrl-P is the prefix character. Any character you enter after Ctrl-P is inserted directly into the document. (This used to be the way you inserted printer codes into a document, but you had to know the codes to insert them.) For example, to put an Escape character into a WordStar document, you press Ctrl-P, and then the Escape key. The DOS 5 Editor also uses this technique. WordPerfect uses Ctrl-V as its prefix key, and other word processors probably incorporate similar schemes.

The Character Code You Can't Enter with the Alt Key

Of the 256 possible character codes, there is one you can't enter with the Alt-key method: character 0, sometimes called the null character because many programs ignore it. (This is not the number 0, which is character code 48; if this seems confusing, check Figure C-1 in Appendix C.)

As the chart in Appendix C shows, no character corresponds to character 0. Sure, it's called Control-@, but pressing Ctrl-@ on your keyboard won't produce character 0. To enter character 0 into a file, press F7 while copying from the console or while using Edlin, and when using the Debug program (described in Chapter 5).

Characters for Drawing Lines and Boxes

Many application programs draw lines or boxes on the screen. The lines and boxes are made up of individual characters whose codes are between 176 and 223. There are four sets of corners and straight lines for drawing with double lines, single lines, and line combinations. You can use these characters to draw your own lines or boxes. Using the numbers on the numeric keypad, type:

```
C><Alt-10><Alt-10>
```

to start on a new line, and type:

```
C><Alt-201><Alt-205><Alt-187>
```

DOS displays the top half of a small box:

```
C>
╔╗
```

Type:

```
C><Alt-10>
```

to move down to the next line, and type:

```
C><Alt-200><Alt-205><Alt-188>
```

DOS completes the box:

```
C>
╔╗
╚╝
```

There are some possibilities here. Press Esc to clear the command line, then press Enter to get back to the system prompt.

The next example should give you some ideas about the many ways you can use these non-keyboard characters. You'll create a small file named MENU.DOC by copying from the console (keyboard) to a file. In the lines to be typed, note that lines 3 and 7 are blank. Further, F1 in line 5 means to type F and 1, not the first function key, and the same is true of F2 in line 9.

Finally, the last line means press Ctrl-Z (or press the F6 function key to accomplish the same thing). Now enter the following:

```
C>copy con menu.doc
Function Keys

<Alt-201><Alt-205><Alt-205><Alt-187>
<Alt-186>F1<Alt-186> Word Processing
<Alt-200><Alt-205><Alt-205><Alt-188>

<Alt-201><Alt-205><Alt-205><Alt-187>
<Alt-186>F2<Alt-186> Communications
<Alt-200><Alt-205><Alt-205><Alt-188>
^Z
```

The screen should look like Figure 2-2.

```
C>copy con menu.doc
Function Keys

╔══╗
║F1║ Word Processing
╚══╝

╔══╗
║F2║ Communications
╚══╝
^Z
```

Figure 2-2. *How DOS echoes the line-drawing and box-drawing characters.*

You can display MENU.DOC, just as you can display any other text file, by using the Type or the Copy command:

```
C>type menu.doc
```

Or:

```
C>copy menu.doc con
```

For better effect, use a batch file so that you can clear the screen first. Enter the following to create a short batch file named SHOWMENU.BAT:

```
C>copy con showmenu.bat
@echo off
cls
type menu.doc
^Z
```

Now test the batch file by typing the following:

```
C>showmenu
```

DOS clears the screen and displays a short function-key menu:

```
Function Keys

F1 Word Processing

F2 Communications
C>
```

These line-drawing and box-drawing characters are used throughout this book. If your word processor or text editor accepts these characters, you can enter them using the Alt-key technique described earlier, then copy, move, or delete them just as you can any other character. As you'll see in Chapter 4, if your printer accepts the extended characters, you can print them, too.

Chapter Summary

- Non-keyboard characters are the characters that can't be typed by pressing a key. Their codes are 0 through 31, and 127 through 255.
- You can enter any character code except 0 (zero) by holding down the Alt key, typing the code number on the numeric keypad, then releasing the Alt key. To enter character 0, press F7 while copying from the console or using Edlin.
- Codes 0 through 31 are defined by the ASCII standard as control characters, such as a carriage return or a tab, not as printable characters. The IBM character set, however, does define printable characters for codes 1 through 31. (See Appendix C.)
- Codes 32 through 127 represent the standard ASCII text characters; codes 128 through 253 represent international characters, line-drawing and box-drawing symbols, and mathematical symbols.
- If your word processor accepts non-keyboard characters such as Alt-205, you can work with them just as you work with standard characters. If your word processor does not accept non-keyboard characters, use Edlin or the DOS 5 Editor to enter them into a file.

CHAPTER

3

Taking Control of Your Screen and Keyboard with ANSI.SYS

Most application programs take pains to make themselves attractive and easy to use. They don't simply display one line after another, scrolling old lines off the top of the screen; they display their messages and results in specific areas of the screen. They also control the color and brightness of what they display, and often assign special meanings to certain keys. With a DOS program called ANSI.SYS, you can use these same techniques to make your batch files and system prompt more attractive and easier to use.

This chapter shows you how to control the display and keyboard with ANSI commands. After trying the examples in this chapter and using the ANSI command summary in Appendix A, you'll be ready to design your own screens. Using a few additional techniques described in other chapters, you can design a menu system tailored to your computer and to the people who use it. Chapter 15 shows one example of such a menu system. It uses batch files and ANSI commands to guide you—or anyone else who uses your computer—through routine use of application programs and batch files.

The ANSI.SYS driver allows you to use ANSI commands with DOS. Note that ANSI.SYS is the name of the device driver—the software. ANSI itself actually refers to a standard for the codes that control your screen and keyboard. You need to load the ANSI.SYS driver to take advantage of ANSI commands.

ANSI.SYS Is a Device Driver

Every device attached to your computer must be controlled by a program. DOS itself controls the disk drives and provides a limited amount of control for the keyboard, display, and printer. For more precise control, a program called a *device driver* is required to tell DOS how to use (or drive) a specific device.

Most device drivers are stored in files with the SYS extension. You might have programs on your system disk such as RAMDRIVE.SYS, which simulates a disk drive in memory; MOUSE.SYS, which controls a computer mouse; or HIMEM.SYS, which manages extended memory.

In order to use a device driver, you must tell DOS about it by adding a Device command to your system's CONFIG.SYS. The Device command names the device driver, and DOS then loads that driver into memory. For example:

```
device=ramdrive.sys
```

Or:

```
device=mouse.sys
```

The ANSI.SYS device driver provides much more detailed control of the console device—the display and keyboard—than DOS does.

The Device command that names ANSI.SYS has two optional parameters:

```
device=ansi.sys /X
device=ansi.sys /K
```

/X lets ANSI.SYS redefine the added keys on an enhanced keyboard. Normally, ANSI.SYS assumes that two keys with the same name, such as the End key on the cursor keypad and the End key on the numeric keypad, are the same. If you want to use these keys for different purposes, then specify the /X switch.

/K tells ANSI.SYS to ignore the keys that appear only on the enhanced keyboard. This makes ANSI.SYS more compatible with older software programs that may incorrectly interpret those keystrokes.

Preparing for the Examples

If you want to take advantage of ANSI.SYS and the ANSI screen and keyboard commands, the CONFIG.SYS file must be in the root directory of your system disk, and it must contain a Device command that tells DOS where to find ANSI.SYS. For example, if ANSI.SYS is in the root directory of drive C, CONFIG.SYS should contain the following:

```
device=c:\ansi.sys
```

If ANSI.SYS is in the DOS subdirectory of drive C, CONFIG.SYS should contain the following:

```
device=c:\dos\ansi.sys
```

If you changed CONFIG.SYS to meet these requirements, restart DOS by pressing Ctrl-Alt-Del before continuing. DOS reads CONFIG.SYS only when it starts, so you must reboot to work with the new version.

ANSI Commands

Using the ANSI display commands, you can position the cursor, tell DOS to display something in high intensity, change colors, or clear the screen. Combining ANSI commands with text gives you near-total control over the appearance of the display.

Using the ANSI Set Keyboard Strings command, you can change the effect of pressing a key, causing a key to produce a different character or even a series of characters, such as your name or a DOS command. This is a "hidden" power of DOS that many users often overlook.

ANSI commands are sometimes called escape sequences because they all begin with the Escape character (ASCII code 27) followed by a left bracket. Most ANSI commands include a numeric or alphabetic code, and each command ends with a different letter. The ANSI.SYS device driver, residing in memory, is constantly looking for escape sequences. When it finds one, it carries out whatever instructions are held within that sequence.

> ❑ **NOTE:** *Throughout this book, the location of an Escape character will be marked by an* [E] *symbol. The method for entering Escape characters varies from program to program and is described in detail in the following sections.*

The general format of an ANSI command is:

[E][*code letter*

The [E][identifies the start of the command, *code* is the guts, and the ending *letter* identifies the command. For example, the command to move the cursor down five rows is:

[E][5B

The code is 5, indicating 5 rows, and the ending letter is B, which moves the cursor down. Some codes contain more than one number or string (a series of characters), separated by semicolons. The command to move the cursor to row 10, column 20, for example, is:

[E][10;20H

The case of the final letter is important. For example, uppercase H identifies the command that moves the cursor, but lowercase h identifies the

command that sets the display mode. Also note that 5, 10, and 20 above, and all values in any ANSI escape sequence, are text characters—not numbers or byte values.

Because ANSI commands control the console device, they must be typed at the keyboard or sent to the display. If you could type the ANSI command ␛[2J at the DOS prompt, DOS would clear the screen. But there's a catch: When you press the Esc key, DOS cancels the line you're typing. The end result would then be "[2J", to which DOS responds, *Bad command or file name*.

One solution is to use the Prompt command. Prompt allows you to enter the Escape character *in code*. So Prompt can be used to send ANSI commands to the screen as well as to change your system prompt. For a quick preview of what you can do with ANSI commands, enter the following Prompt command:

```
C>prompt $e[7m$e[2J
```

You should be looking at a blank screen in reverse video; if not, reenter the Prompt command again. The system prompt you just defined is actually two ANSI commands: $e[7m sets the display to reverse video, and $e[2J clears the screen.

Note that there is no system prompt. You defined the prompt to be the ANSI commands, which affect the appearance of the screen but don't actually display a prompt. Enter the following commands to restore the normal screen and standard DOS system prompt (the second Prompt command restores the standard system prompt):

```
prompt $e[m
prompt
C>cls
```

Your screen should be back to normal.

Sending ANSI commands to the console

You can't type an ANSI command because all ANSI commands start with an Escape character, and DOS cancels the command line when you press the Escape key. So to get ANSI commands sent to the display you have three alternatives:

- Use the Prompt command to send the ANSI commands to the console.
- Put the ANSI commands in a file and display the file with the Type or Copy command.

- Put the ANSI commands in a batch file and use Echo commands to send them to the console. (See Chapter 7 for examples of this method.)

Creating a file that contains ANSI commands

Creating a file that contains ANSI commands isn't difficult. Some text editors (such as Editor or Edlin) and word processors (such as Microsoft Word) let you enter an Escape character. You could also use the DOS Debug program, described in Chapter 5. Be sure the program you use also lets you save a file without saving the formatting codes (such as the Text Only format in Microsoft Word, the non-document mode in WordStar, or the DOS text mode in WordPerfect).

Once you have created the file and stored it on disk, carry out the ANSI commands in the file by displaying it with the Type or Copy command. This technique is best for all but the simplest use of ANSI commands. It's used in this chapter and elsewhere in the book for the longer examples.

Using the Prompt Command to Carry Out ANSI Commands

For simpler uses of ANSI commands, you can enter an ANSI command without using a text editor or a word processor: The Prompt command tells DOS what to display for the system prompt, and it lets you specify an Escape character without pressing the Escape key. You can include an ANSI command in the system prompt, or even make an ANSI command the entire system prompt.

A brief review of the Prompt command

To let you tailor your system prompt, the Prompt command accepts special codes beginning with the dollar sign. The current drive, for example, is represented by $n, and the greater-than sign (>) is represented by $g, so you would specify the standard system prompt (C>) with:

```
prompt $n$g
```

Figure 3-1 lists all the special dollar sign codes you can use with the Prompt command, and what each of them does.

What does all this have to do with ANSI commands? One of the special codes you can specify with the Prompt command—$e—represents the Escape character. This is the troublesome character that DOS won't allow you to type. So, by using $e with the Prompt command, you can send an ANSI command to the display.

Prompt Command	*Displays*
$$	$, dollar sign character
$b	|, pipe character
$d	the date (according to the system clock)
$e	the Escape character
$g	>, greater-than character
$h	backspace (erase previous character)
$l	<, less-than character
$n	the logged disk drive letter
$p	the logged disk drive and subdirectory
$q	=, equal-sign character
$t	the current time (according to the system clock)
$v	DOS version
$_	Carriage return/line feed (new line)

Figure 3-1. *The special Prompt commands.*

Preparing for the examples

The short examples in this chapter use a Prompt command that defines the system prompt as an ANSI command. Most of the examples result in no visible system prompt. If you wish, you can restore the standard prompt after each example by typing "prompt" alone and pressing Enter.

If you have defined your own system prompt—perhaps with a Prompt command in your AUTOEXEC.BAT file—you might want to create a batch file that contains the Prompt command that defines your system prompt. That way, you can restore your system prompt simply by typing the name of the batch file after each example.

The following is a sample batch file that will restore a system prompt showing the current drive and path. Replace this Prompt command with one that restores your own favorite system prompt if different.

To create this batch file and name it PROMPTRS.BAT, enter the following, using your own prompt command if different:

```
C>copy con promptrs.bat
prompt $p$g
^Z
```

Now, to restore your system prompt, type *promptrs* after entering any of the following sample Prompt commands that demonstrate ANSI.SYS.

Controlling the Display

The ANSI commands for controlling the display fall into three groups:

- *Cursor control* commands, which let you move the cursor up, down, left, right, or to a specific position; save the current cursor position; and restore the cursor to the saved position.
- *Erase* commands, which let you erase the entire display or just part of a line.
- *Attribute and mode* commands, which let you control the number of columns (40 or 80), graphics and color modes, and attributes such as high intensity, blinking, foreground color, and background color.

Appendix A contains a complete summary of ANSI commands, with descriptions of each command and brief examples of the most common uses. This chapter shows you how to use a few of the ANSI commands; several other commands are used in other chapters of this book. The general form of each command is the same, so after finishing the examples here you will be able to use any commands described in Appendix A.

The Set Graphics Mode command

The Set Graphics Mode command lets you control display attributes such as high intensity, blinking, and color. Because color attributes (other than black and white) are simulated on a monochrome display, some combinations of background and foreground colors are difficult to read. Some combinations produce undesirable effects even on a color display. (For example, black text on a black background has always been most difficult to read.) You may want to experiment a bit to find a combination that is both pleasing and easy to see.

A Set Graphics Mode command affects everything that is displayed after it is entered. Figure 3-2 shows a complete list of the attributes and colors.

The Set Graphics Mode command has one parameter, the attribute code:

E[*attr*m

attr is a number that specifies the attribute to be turned on. If you omit *attr*, all attributes are turned off. You can include more than one *attr* at a time, but separate them by semicolons. Figure 3-2 also shows the display attributes and the corresponding values for *attr*.

m is the ending letter that identifies the Set Graphics Mode command.

Non-Color	*Attributes*
0	All attributes off
1	High intensity (bold)
4	Underline (monochrome display only)
5	Blink
7	Reverse video
8	Invisible

Foreground Color		*Background Color*	
30	Black	40	Black
31	Red	41	Red
32	Green	42	Green
33	Yellow	43	Yellow
34	Blue	44	Blue
35	Magenta	45	Magenta
36	Cyan	46	Cyan
37	White	47	White

Figure 3-2. *Display-attribute codes.*

Changing the display to reverse video

The screen normally displays light characters on a dark background. Displaying dark characters on a light background is called *reverse video*. As Figure 3-2 shows, the attribute code for reverse video is 7. To change your display to reverse video, type the following Prompt command to send a Set Graphics Mode command with the attribute code 7 to the display:

```
C>prompt $e[7m
```

Remember, all ANSI commands begin with an Escape character followed by a left bracket and end with a letter. The parameter for the Prompt command begins with $e, which represents the Escape character, followed by a left bracket. The 7 is the attribute code for reverse video, and m is the ending letter that identifies the Set Attribute command. Each time DOS would normally display the system prompt, it sends this Set Attribute command to the display.

Because of all the brackets and numbers, an ANSI command can get pretty hard to read, so you may often make a mistake when you type one. If any part of the ANSI command you just typed is displayed, you probably made a typing error; if 7m is displayed, for example, you probably forgot the $e or the left bracket. Just reenter the Prompt command.

If nothing seems to happen, everything's OK. There's no system prompt on the line that contains the cursor because you defined the prompt to be just an ANSI command. Clear the screen by entering:

```
cls
```

The screen has changed to reverse video (dark characters on a light background). But there's still no prompt, because the Prompt command didn't specify anything to be displayed (you'll restore the prompt in a moment). To put something on the screen, enter a Directory command:

```
dir
```

A list of files is displayed in reverse video. To change back, enter another Set Graphics Mode command that specifies no attribute code. This turns off all attributes, restoring the display to normal. Enter this Prompt command:

```
prompt $e[m
```

Again, if any part of the Prompt command is displayed, enter it again. Now restore the system prompt by entering either a Prompt command with no parameters or the name of a batch command that restores your system prompt:

```
promptrs
```

Remember, you can always restore the standard system prompt by just entering the following:

```
prompt
```

Notice that what you typed is displayed normally (light characters on a dark background), but the rest of the screen still shows a light background. Restore the entire screen by entering another Clear Screen command:

```
C>cls
```

The Cursor Position command

The Cursor Position command moves the cursor to a specific row and column. Anything displayed after a Cursor Position command starts at the new cursor location.

The Cursor Position command has two parameters:

E[*row*;*col*H

The *row* parameter specifies the row to which the cursor is to be moved. It can be any number from 1 through 50, depending on which display you have. If you omit *row*, DOS assumes 1; if you omit *row* and want to include *col*, type the semicolon so that DOS will know that *row* is missing.

col specifies the column to which the cursor is to be moved. It can be any number from 1 through 80. If you omit *col*, DOS assumes 1.

H is the ending letter that identifies the Cursor Position command.

Displaying something at a specific location

Suppose you want to display "Middle of the screen" at row 12, column 30. Enter the following Prompt command to send a Move Cursor command to the display:

```
C>prompt $e[12;30HMiddle of the screen
```

Middle of the screen should be displayed near the middle of the screen. If it isn't, reenter the Prompt command.

Restore the system prompt by entering *prompt* or the batch command (*promptrs*) that restores your system prompt. The command you type will follow the word *screen* because that's the end of the current system prompt.

Enter *cls* to clear the screen.

Displaying at a specific location with attributes

You can combine the features of the first two examples by specifying both a specific location and display attributes. For example, you could display the words *Bold Blink Normal* starting at row 15, column 1, in attributes that match the words. If you have a color monitor, you can change the color of each word, too.

Two versions of this example are shown, one for monochrome monitors and one for color. Follow the instructions under the heading that's appropriate for your monitor, and then go on to the heading "Did It Work?" Each example includes four ANSI commands in a single Prompt command, so type carefully; guard especially against leaving out one of the left brackets. As before, if part of the command is displayed, you'll have to type it all again.

If you have a monochrome monitor

If you have a monochrome monitor, you'll send the following ANSI commands to the display:

Command	*Description*
$e[15H	Moves the cursor to row 15, column 1 (the column parameter is omitted, so DOS assumes 1).
$e[1mBold	Sets attribute 1 (high intensity) and displays the word *Bold*.
$e[5mBlink	Sets attribute 5 (blink) and displays the word *Blink*.
$e[mNormal	Turns all attributes off and displays the word *Normal*.

Enter the following Prompt command:

```
C>prompt $e[15H$e[1mBold $e[5mBlink $e[mNormal
```

Now go to the heading "Did It Work?"

If you have a color monitor

If you have a color monitor, you'll send the following ANSI commands to the display:

Command	*Description*
$e[15H	Moves the cursor to row 15, column 1 (the column parameter is omitted, so DOS assumes 1).
$e[1;31mBold	Sets attributes 1 (high intensity) and 31 (red foreground), and displays the word *Bold*.
$e[5;32mBlink	Sets attributes 5 (blink) and 32 (green foreground), and displays the word *Blink*.
$e[mNormal	Turns all attributes off and displays the word *Normal*.

Enter the following Prompt command:

```
C>prompt $e[15H$e[1;31mBold $e[5;32mBlink $e[mNormal
```

Did it work?

The word *Bold* should be bold (high intensity), *Blink* should be blinking and high intensity, and *Normal* should be in standard letters. If you don't see *Bold Blink Normal* in the intended attributes and colors, retype the Prompt command.

Notice that turning on an attribute doesn't turn off any existing attributes, so *Blink* is both blinking and high intensity. If you didn't want *Blink* to appear in high intensity, you would have to turn off all attributes with code 0, then turn on blink with code 5. You can mix ANSI commands anywhere in text, changing attributes or color or both between each character.

Restore the system prompt by entering *prompt* or the batch command (*promptrs*) that restores your system prompt.

In addition to quick demonstrations of ANSI commands, you can use the Prompt command and ANSI commands to define quite elaborate system prompts, complete with reminders of the current drive, directory, time, and date, all boxed in colored windows if you like. Chapter 10 shows several examples of system prompts that go a bit further than the usual C>.

Putting ANSI Commands in a File

The previous example really stretched the use of the Prompt command to enter ANSI commands. When you need to enter this many commands, it's usually better to put them in a file and display the file with the Type or Copy command.

Combining ANSI commands with text, as in the previous example, lets you design your own screens. For example, suppose your office prepares a monthly report. The procedure is fairly simple, but it happens only once a month, so people sometimes forget how to do it. You decide you need a help screen that anyone can quickly display to get instructions. The help screen tells the user that preparing the report requires three steps:

1. Enter the monthly data into a spreadsheet.
2. Update a word-processing document named REPORT.DOC.
3. Type the command *doreport*.

In addition, you want to remind the user that all printer output should be sent to the department office.

Using DOS 5's Editor—or any other editing program that allows you to enter the Escape character—you can put these instructions, plus a few ANSI commands, in a file named MONTHRPT.HLP that will display the help screen. To see the help screen, simply display the file with the Type or Copy command.

Entering the Escape character in a file

The next example requires you to enter a series of ANSI commands in a file using a word processor or text editor that can save a file without formatting codes. (Most word processors save text in a formatted file. To use ANSI commands with DOS, the file must be saved as ASCII, plain text, or whatever term your application uses for an unformatted file.)

Entering an Escape character into your word processor is sometimes as difficult as doing it in DOS, but it is possible. Some word processors, such

as Microsoft Word, allow you to enter nontext characters such as the Escape character by using the Alt-key-and-numeric-keypad technique described in Chapter 2. Other programs may offer different techniques. For example, in Editor and WordStar (plus all other WordStar-like programs), you press Ctrl-P and then Escape. In WordPerfect, you press Ctrl-V and then Escape. Check the manual for your personal favorite to see whether you can enter these characters and if so, how. If you don't have such a program, you can use Editor or Edlin. For a description of how to use either of those editors, see your DOS manual or *Running MS-DOS*.

What your program displays when you enter the Escape character depends on the software. If you look back to Appendix C, you can see that the IBM-defined symbol for the Escape character is a left arrow (←). Microsoft Word, WordStar, and Editor display that symbol. WordPerfect displays Control-left bracket (^[), which is the accepted way to represent Escape as a control character.

If you're using Edlin, you type Ctrl-V and then a left bracket to enter an Escape character. Edlin displays this as ^V[, its way of representing the Escape character. Note that this left bracket is not the same left bracket that follows the Escape character at the beginning of an ANSI command; you must type a second left bracket, then the rest of the ANSI command. So in Edlin, all ANSI commands will look like ^V[[when you enter them. When you list the lines, however, they'll look like this: ^[[.

Different versions of the next example are shown for monochrome and color monitors. Follow the instructions appropriate for your monitor.

If you have a monochrome monitor

Use your editing program to create a file named MONTHRPT.HLP, and enter the lines shown in Figure 3-3. The line numbers shown there are for reference only; start each line with the first Escape character.

```
1: ←[2J
2: ←[5;10H←[1mPREPARING THE MONTHLY REPORT
3: ←[10;20H←[m1. Enter data into spreadsheet
4: ←[12;20H2. Update WP document named REPORT.DOC
5: ←[14;20H3. Type ←[1mdoreport
6: ←[20;10H←[7mSend printer output to department office
7: ←[m
```

Figure 3-3. *Monochrome help-screen text and commands.*

The following explanations of the ANSI commands refer to the line numbers shown in Figure 3-3:

1. [2J clears the screen.
2. [5;10H moves the cursor to row 5, column 10. [1m turns on the high-intensity attribute.
3. [10;20H moves the cursor to row 10, column 20. [m turns off all display attributes.
4. [12;20H moves the cursor to row 12, column 20.
5. [14;20H moves the cursor to row 14, column 20. [1m turns on the high-intensity attribute.
6. [20;10H moves the cursor to row 20, column 10. [7m turns on the reverse-video attribute.
7. [m turns off all display attributes.

Proofread the file carefully, save it, and go on to the heading "Displaying the help screen."

If you have a color monitor

Use your editing program to create a file named MONTHRPT.HLP and enter the lines shown in Figure 3-4. The line numbers shown there are for reference only; start each line with an Escape character.

```
1: [46m
2: [2J
3: [5;10H[31mPREPARING THE MONTHLY REPORT
4: [10;20H[34m1. Enter data into spreadsheet
5: [12;20H2. Update WP document named REPORT.DOC
6: [14;20H3. Type [1mdoreport
7: [20;10H[33;41mSend printer output to department office
8: [m
```

Figure 3-4. *Color help-screen text and commands.*

The following explanations of the ANSI commands refer to the line numbers shown in Figure 3-4:

1. [46m sets the background color to cyan.
2. [2J clears the screen.
3. [5;10H moves the cursor to row 5, column 10. [31m sets the foreground color to red.
4. [10;20H moves the cursor to row 10, column 20. [34m sets the foreground color to blue.
5. [12;20H moves the cursor to row 12, column 20.

6. ⎋[14;20H moves the cursor to row 14, column 20. ⎋[1m turns on the high-intensity attribute.
7. ⎋[20;10H moves the cursor to row 20, column 10. ⎋[33;41m sets the foreground color to yellow and the background color to red.
8. ⎋[m turns off all display attributes.

Proofread the file carefully and save it. Be sure to save the file without any formatting codes if you're using a word processor.

Displaying the help screen

To display the help screen, type:

```
C>type monthrpt.hlp
```

Your screen should look like Figure 3-5. If it does not, load MONTHRPT.HLP with your editing program and proofread it against the instructions. Then correct any errors, save the file, and enter the Type command again.

If you have a monochrome display, the title should be in high intensity, the numbered items normal, *doreport* in high intensity, and the bottom line (*Send printer output...*) in reverse video.

```
PREPARING THE MONTHLY REPORT


          1. Enter data into spreadsheet

          2. Update WP document named REPORT.DOC

          3. Type doreport


          Send printer output to department office

C>
```

Figure 3-5. *Help screen.*

If you have a color display, the background should be cyan, the title in red, the numbered items in blue, *doreport* in high-intensity blue, and the bottom line (*Send printer output...*) in high-intensity yellow on a red background.

To display this help screen more quickly, you can put the Type command in a batch file named MONTH.BAT; anyone can then display the help screen by typing *month*. As you'll see in a moment, you can make it even easier to display a help screen by using the ANSI command that assigns a string (series of characters) to a key.

Controlling the Keyboard

All the examples thus far have dealt with controlling where and how text is displayed, or the output of the console device. ANSI.SYS has one other major capability that deals with the keyboard, or the input of the console device: You can change the characters that are produced when a key is pressed.

The Set Keyboard Strings command

The Set Keyboard Strings command lets you specify one or more characters to be produced when a key is pressed. You can redefine almost every key on the keyboard, including the function keys. You can even redefine combinations of the Shift, Ctrl, and Alt keys with most other keys.

The Set Keyboard Strings command has two parameters: the identification code of the key and the characters to be produced when the key is pressed:

E[*key code*;*result* p

key code is a number that identifies the key you're defining. For the main, typewriter, portion of the keyboard, *key code* is the same as the ASCII code for that character. Unshifted k, for example, is 107, and shifted K is 75; Appendix D contains the complete chart of key codes for the Set Keyboard Strings command. As you'll see in a moment, you can use two numbers separated by a semicolon to specify other keys, such as the function keys, numeric keypad, and combinations of keys.

result is the character or characters to be produced when the key specified in *key code* is pressed. *result* can be any combination of ASCII codes and strings enclosed in quotation marks, separated by semicolons. All the characters in *result* are produced each time the key is pressed.

To restore a key's original meaning, specify a *result* identical to the *key code*.

p is the ending letter that identifies the Set Keyboard Strings command. Note that it's a lowercase p.

Redefining a key

Because the Set Keyboard Strings examples are fairly short, you'll use the Prompt command to enter them, just as you did earlier in this chapter.

Suppose you don't use the reverse apostrophe (`) key, but you would like to be able to type the fraction ½ directly from the keyboard. The key code for the reverse apostrophe key is 96, the same as its ASCII code; the character code for the fraction ½ is 171. (You can find both these numbers in the table in Appendix C.)

To make the reverse apostrophe key produce ½, type the following Prompt command:

```
C>prompt $e[96;171p
```

Now press the key labeled with the reverse apostrophe. (On an IBM enhanced keyboard, it's at the upper left of the typewriter portion of the keyboard, above the tab key.) DOS should display ½. This redefinition stays in effect until you restart DOS.

If you wished, you could redefine any or all of the keys to include graphics characters or characters from other languages. You could even arrange the entire keyboard in some other layout, such as the Dvorak keyboard. By putting the Set Keyboard Strings commands into a batch file, and by putting another set of Set Keyboard Strings commands (to restore the original keyboard) into a second batch file, you could redefine or restore your keyboard by typing a single command.

To restore the original meaning of the reverse apostrophe key, you enter a Set Keyboard Strings command that specifies a result code identical to the key code. As you saw in the previous example, the key code and the ASCII character code for the reverse apostrophe are both 96, so both numbers in the Set Keyboard Strings command should be 96. Type the following Prompt command:

```
C>prompt $e[96;96p
```

Press the reverse apostrophe key again; DOS should display the reverse apostrophe. Now restore the system prompt by typing *prompt* or the batch command that restores your system prompt.

This is a neat trick with a potential for lots of use. But note that your redefinitions will work only in DOS and those few programs that use DOS's ANSI.SYS driver. Although pressing the reverse apostrophe key will display a fraction at the DOS prompt, it probably won't work in your word processor or any other major application.

Using the extended key codes

As the previous example showed, the key code for a standard typewriter keyboard character is the same as the ASCII code for the character. But what about the rest of the keyboard—the function keys and direction keys, the combinations of the Shift, Ctrl, and Alt keys with other keys? You specify these keys by entering an *extended code,* which is two numbers separated by a semicolon. The first number of the extended code is always 0; the second number is an identification code that ANSI assigns to each key or combination of keys that can be redefined. Appendix D gives the extended key code for each key and combination of keys (on an IBM or IBM-compatible keyboard) that you can redefine with ANSI.

Function key F1, for example, is represented by 0;59, Alt-A by 0;30, and the right-arrow key by 0;77. Combinations that cannot be redefined—such as Ctrl-Up Arrow—are noted in Appendix D. If a key doesn't appear in Appendix D at all, it cannot be redefined.

Assigning a command to a key

If you type the same command fairly often, you can save yourself time by assigning the entire command to a single key. Suppose you frequently find yourself typing *dir ¦ sort*; you can assign that string to a key by entering the string, enclosed in quotation marks, in the Set Keyboard Strings command. Because this is a kind of Directory command, assign it to the combination of the Alt key and D (referred to as Alt-D). The identification code for Alt-D is 0;32, so type the following Prompt command:

```
C>prompt $e[0;32;"dir ¦ sort";13p
```

Hold down the Alt key and press D; DOS should display the Directory and Sort commands, then carry them out. As far as DOS is concerned, each time you press Alt-D it's just as if you typed *dir ¦ sort*.

In the Set Keyboard Strings command you just typed, notice that a semicolon and the number 13 follow the string in quotation marks. Because 13 is the ASCII code for a carriage return, including it in the result code has the same effect as pressing the Enter key. If you didn't include the 13, DOS would display the string and leave the cursor at the end of the line. You would have to press Enter to carry out the command.

To restore Alt-D to its original meaning, type the following Prompt command:

```
C>prompt $e[0;32;0;32p
```

Now try pressing Alt-D again; nothing should happen. Restore the system prompt by typing either *prompt* or the batch command that restores your system prompt, *promptrs*.

Displaying the help screen with one keystroke

Earlier in this chapter, you created the file called MONTHRPT.HLP, which contained instructions for preparing a monthly report, and then displayed the file with the Type command. Now you can use the Set Keyboard Strings command to display the help screen with a single keystroke by assigning the Type command to a key.

Suppose you decide that Shift-F1 should display the help screen. Type the following Prompt command:

```
C>prompt $e[0;84;"type monthrpt.hlp";13p
```

Now hold down either Shift key and press the F1 key. DOS should display the help file just as it did before.

To restore Shift-F1 to its original meaning, type the following:

```
C>prompt $e[0;84;0;84p
```

Press Shift-F1 again; nothing should happen. Restore the system prompt by typing *prompt* or the batch command that restores your system prompt.

Using this technique, along with others presented in this chapter, you could create a whole series of help screens (or other screens tailored to your system) and display each simply by pressing a single key. A word of caution: Be sure not to redefine a key that has special meaning to DOS. If you redefine Ctrl-C (ASCII code 3), for example, its original meaning is lost and you won't be able to cancel a command unless you restore the original meaning. If you redefine the Enter key (ASCII code 13), you won't be able to enter another command to undo the redefinition, and you'll have to re-start DOS to do so. To be on the safe side, don't redefine any key whose key code is less than 32 unless you're confident that you know what the result will be.

Also, keep in mind that many application programs assign their own meanings to keys, and these redefinitions may override your redefinitions. This usually isn't a problem because you can't enter DOS commands while

you're using the other program, and your redefinitions become effective again when you leave the program and return to DOS.

Chapter Summary

- ANSI.SYS is a device driver that lets you control the display and the keyboard.
- To use ANSI commands, the files CONFIG.SYS and ANSI.SYS must be on your system disk. CONFIG.SYS must be in the root directory and must contain a Device command that tells DOS where to find ANSI.SYS.
- The ANSI display commands let you control the cursor, set the display mode and attributes, and erase part or all of the screen.
- The ANSI Set Keyboard Strings command lets you redefine the effect of a key.
- All ANSI commands start with [Esc][and end with a letter. Most also include a code (one or more numbers or strings).
- ANSI commands must be sent to the display. Because you can't type the Escape character, you must either put the commands in a file and display the file, or use the Prompt command.
- Appendix A shows a summary of all ANSI commands. Appendix C shows the full ASCII and IBM-compatible character sets. Appendix D shows the extended codes for all non-ASCII keys and combinations.

CHAPTER

4

Learning Your Printer's Language

If you use a word processor, you use its commands to specify when you want something printed in boldface or italics, to change the margins, or to force the start of a new page. When you print a document, the word-processing program sends the necessary commands to the printer. If you use a graphics program, you use its commands to describe the chart or the image, and the program sends the necessary commands to the printer. You never have to "talk" directly to the printer at all.

But you can control your printer without using one of these application programs. This chapter shows you how to control the way a file is printed on an Epson or Epson-compatible dot-matrix printer or a Hewlett-Packard or HP-compatible laser printer by sending commands to the printer yourself. This chapter also introduces you to the printer commands and shows you how to use a few of them. Chapter 12 shows several ways you can use your printer to do more than simply print word-processing documents and spreadsheets and describes more printer commands. Appendix B is a quick reference to the more commonly used commands of Epson-compatible and HP-compatible printers.

You'll also learn in this chapter whether your printer can use the IBM extended character set, shown in Appendix C.

Printer Commands

Like the ANSI.SYS commands described in Chapter 3, printer commands aren't words but consist of letters, numbers, and non-keyboard characters. Also like the ANSI.SYS commands, all but a few printer commands begin with the Escape character, which means you can't type most of these commands directly from DOS.

Printer commands fall into four general categories:

- *Typeface* commands control the type of print to be used, which may include normal, bold, extended, compressed, elite, and italic, and on some of the advanced printers will change the font or character set printed.
- *Format* commands control such characteristics as the space between lines, skipping to a new line or page, page size, margins, and tabs.
- *Graphics* commands control the printing of graphics images.
- *Printer control* commands reset the printer to its standard settings, sound the bell, cause the paper-out detector to be ignored, set the direction and speed of printing, and control other machine-related functions.

Printer manufacturers are free to design their own commands, but most laser printers available today use the same commands and character set as Hewlett-Packard laser printers, just as most dot-matrix printers use the Epson or IBM printer commands. This chapter shows you how to use some of the commands common to those types of printers, but it doesn't cover all the commands; there are simply too many, and most of the commands are seldom needed or used only for advanced printing applications.

Sending Commands to the Printer

You've probably already used DOS commands that communicate with your printer. For example, when you use a Mode command to print small characters (*mode lpt1:132* or *mode lpt1 cols=132* with DOS 5), you send a command to the printer. If that printer is compatible with the early IBM/Epson models, everything printed after that command will be printed in compressed type. There are more direct ways, however, to send commands to the printer. In the next few pages, for example, you will:

- Redirect an Echo command to the printer. This is a quick way to enter short commands that don't begin with an Escape character.
- Copy from the console to the printer. This is another way to enter a few commands that don't require the Escape character.
- Copy a text file to the printer. This is the best way to enter a longer series of commands, especially those with the required Escape character.

The Hewlett-Packard laser printers, and those compatible with its special command language, PCL, almost without exception use the Escape character to start a command. Because of this, the following few sections show how to use only Epson printing commands. Commands specific to HP and compatible laser printers are covered in the section "Hewlett-Packard PCL printer commands."

Sending printer commands with the Echo command

The Echo command is most commonly used in batch files to display messages, but you can also enter it at the system prompt like any other command. If you redirect the output of the Echo command to the printer, you can use it to print a line or to send a command to the printer. Output is redirected to the first parallel printer port (lpt1) by using the DOS redirection symbol (>). To send the output to a serial port, you must first use the Mode command to redirect the parallel port output to the serial port.

It's easy to test the Echo command. The printer command for compressed (small) print on Epson and Epson-compatible printers is character 15. This

doesn't correspond to any key on the keyboard, but you can get it by typing Ctrl-O. Make sure your printer is turned on, and type the following Echo command (DOS echoes Ctrl-O as ^O):

```
C>echo <Ctrl-O>This is small. > prn
```

When you press Enter, the line should be printed in compressed type. You could have entered the compressed-print command by holding down the Alt key and typing the numbers 1 and 5 on the numeric keypad (referred to as Alt-15, and described in Chapter 2), but Ctrl-O is quicker to type, and DOS echoes the character as ^O no matter how you enter it.

After you enter a command for compressed print, the printer continues to use compressed print until you enter a command to stop compressed print or reset the printer. To see how this works, print another line by entering:

```
C>echo This line, too, is small. > prn
```

The printer is still using compressed print.

You can enter a print command anywhere in a line. The command to stop compressed print is character 18, or Ctrl-R. Enter:

```
C>echo <Ctrl-R>This is a<Ctrl-O> small<Ctrl-R> word. > prn
```

The result should look like this:

```
This is a small word.
```

The command to start expanded (large) print is character 14, or Ctrl-N, and the command to stop expanded print is character 20, or Ctrl-T. Enter:

```
C>echo This is a<Ctrl-N> large<Ctrl-T> word. > prn
```

The word *large* should be in expanded type:

```
This is a  L a r g e  word.
```

The printer cancels the command to start expanded print after one line because it assumes that you would use the large letters only for one-line headings. Enter these two Echo commands:

```
C>echo <Ctrl-N>This line is large. > prn
C>echo This line is normal. > prn
```

The second line is normal type, although you didn't enter a command to stop expanded print; the printer returned to normal type after the first line:

```
This line is large.
This line is normal.
```

You can also combine printer commands. Combining compressed and expanded print yields a darker print, similar to the bold print you'll use in a moment, but somewhat wider. Enter the following Echo command:

```
C>echo <Ctrl-N><Ctrl-O>This is both expanded and small. <Ctrl-
R> > prn
```

You entered Ctrl-R at the end of the line because compressed print doesn't turn itself off as expanded print does.

Sending printer commands by copying from the console

This example lets you see the effect of the commands you have used so far. Enter:

```
C>copy con prn
This is normal.
<Ctrl-N> This is expanded.
<Ctrl-O> This is small.
<Ctrl-N> This is small and expanded.
<Ctrl-R> This is normal.
^Z
```

You should see five lines that look like this:

```
This is normal.
This is expanded.
This is small.
This is small and expanded.
This is normal.
```

These two techniques—redirecting the Echo command and copying from the console to the printer—work only for commands that don't begin with the Escape character because, as Chapter 3 explained, you can't type the Escape character. You have just used the most common printer commands that don't begin with Escape. Figure 4-1 lists these commands.

Command	*Result*
Ctrl-N	Expanded print (one line only)
Ctrl-O	Start compressed print
Ctrl-R	Stop compressed print
Ctrl-T	Stop expanded print

Figure 4-1. *Some Epson-compatible printer commands that don't begin with Escape.*

Putting printer commands in a file

Most of the remaining Epson printer commands (and all Hewlett-Packard PCL commands) start with the Escape character. You must put them in a file using a word processor or text editor that lets you enter non-text characters.

Figure 4-2 shows some of the Epson-compatible printer commands that begin with the Escape character; Figure 4-3 shows corresponding HP PCL commands that perform the same functions, plus the corresponding commands from Figure 4-1.

Command	*Result*
E@	Reset the printer to its standard settings
EG	Start double-strike printing (slightly darker)
EH	Stop double-strike printing
E-1	Start underlining
E-0	Stop underlining
E4	Start italics
E5	Stop italics

Figure 4-2. *Some Epson-compatible printer commands that begin with Escape.*

Hewlett-Packard PCL printer commands

When the first Epson dot-matrix printers were introduced, printers weren't capable of much. Therefore, all their commands could be performed using simple control characters, as seen in Figure 4-1. However, as the printer's abilities expanded, the Escape character was introduced as a prefix for commands controlling the expanded abilities. Originally, these were only two-character commands, but as the printer's abilities grew, the length of the commands grew as well.

A laser printer can do much more than its dot-matrix counterpart. Originally, the HP LaserJet printer was only a fancier (and quieter) form of letter-quality printer, like a daisy-wheel printer. Eventually, more complex commands were added to control such features as 300-dot-per-inch graphics, scalable fonts, typestyles, and downloadable fonts. The printer language that controls all those features is PCL. Over the years it's grown quite complex.

The typical PCL command starts with Escape, followed by one or more special characters, a number, and finally a capital letter, which ends the command. Of course, the LaserJet uses some standard control-character combinations, including the carriage return and line feed and tab. One important control character is the formfeed, Ctrl-L. Unlike a dot-matrix printer, a laser printer doesn't print until it has received either a full page

of text or the formfeed character. (You can also press the Form Feed button on the printer's front panel to eject a page.)

Given these comparative limitations, it's obvious that HP printer commands must be contained in a file. But once they're there, the limitations disappear and you can do amazing things with your printer, with output far better than anything a dot-matrix printer is capable of.

Figure 4-3 contains a list of some of the HP PCL commands, ones that are similar to the Epson printer commands already discussed. Note how each of them starts with the Escape character, which is followed by special characters, a number, and then a capital letter. The default font for the HP LaserJet family of printers is 12-point Courier. Depending on the model of LaserJet you own and the types of downloadable fonts and font cartridges you have installed, you'll also be able to print in a variety of additional fonts, sizes, and type styles. Consult your printer documentation for the specific set of escape sequences you have access to.

Command	*Result*
⎋E	Reset the printer, clear buffer
⎋(s3B	Start bold printing
⎋(s-0B	Stop bold printing (print medium)
⎋&d0D	Start underlining
⎋&d@	Stop underlining
⎋(s1S	Start italics
⎋(s0S	Stop italics (print upright)

Figure 4-3. *Some HP PCL printer commands.*

Figure 4-4 shows a text file, PCLTEST.DOC. Enter this file using your word processor, text editor, Editor, or Edlin. (See the section "Entering the

```
1:  ⎋E
2:  Welcome to your HP PCL LaserJet testing program.
3:
4:  ⎋(s3BThis is basic bold printing.
5:  ⎋(s0BTo stop bold printing, you issue the print medium command.
6:
7:  Underlining words is done with the ⎋&d0Dunderline⎋&d@ command.
8:
9:  ⎋(s1SAnd italic is done like this.⎋(s0S
10:
11: Finally a formfeed kicks this page from the printer:<Alt-12>
```

Figure 4-4. *PCLTEST.DOC, the HP LaserJet testing program.*

Escape Character in a File" in Chapter 3 for a description of how to enter the Escape character.) Don't type the line numbers; they're only for reference.

Here are some tips for entering the file:

- Lines 3, 6, 8, and 10 are blank. Line 2 contains only text.
- Line 1 resets the printer with the ⎋E command, which is very similar to ⎋@ on the Epson printer. This command may also eject a page from the printer, printing any data remaining in the printer's internal buffer.
- Line 4 contains the Stroke Weight command to turn on bold printing. The command is ⎋(s*value*B, where *value* ranges from –7 for ultrathin to 7 for ultrablack. A *value* of 3 is standard bold, and –3 is light text, with 0 being medium. Line 5 starts with a Stroke Weight command and a *value* of 0, to return printing to normal. (Note that older LaserJets do not have the ability to print in bold without a font cartridge or downloadable font installed.)
- Line 7 contains the underline on and underline off commands, surrounding the word *underline*. Unlike other PCL commands, the underline commands contain no values. (There is an additional underline command that allows you to set the position of the underline relative to the characters printed.)
- Line 9 contains commands to start and stop italic printing. (Note that older LaserJets do not have the ability to print in italics without a font cartridge or downloadable font installed.)
- Line 11 is important because it ends with a formfeed character, Ctrl-L. This is necessary to eject the page from the printer when you're done. Enter Ctrl-L using the same techniques you use to enter Escape (<Alt-12>; Ctrl-P, Ctrl-L, and so on).

Finish editing the file, making sure it matches Figure 4-4, then save it to disk as PCLTEST.DOC. Now print the file by entering the command *copy pcltest.doc prn* at the DOS prompt. Figure 4-5 shows the output from an HP LaserJet III printer.

```
Welcome to your HP LaserJet PCL testing program.

This is basic bold printing.
To stop bold printing, you issue the print medium command.

Underlining words is done with the underline command.

And italic is done like this.

Finally, a formfeed kicks this page from the printer:
```

Figure 4-5. *This is the output of PCLTEST.DOC on an HP LaserJet III printer.*

Printing the IBM Extended Character Set

All printers can print the standard ASCII character set (codes 32 through 126). Many printers can also print the IBM extended character set (codes 128 through 255) shown in Appendix C. The sample chart in Figure 4-6 uses eight characters from the extended set to draw a box.

Code	*Character*	*Code*	*Character*
Alt-201	╔	Alt-187	╗
Alt-204	╠	Alt-185	╣
Alt-200	╚	Alt-188	╝
Alt-205	═	Alt-186	║

Figure 4-6. *Box-drawing characters.*

Creating the sample chart file

Suppose you want to convince someone that using a dot-matrix printer to print simple graphics figures would save time and money compared to having them done by a graphic artist. The following example uses the printer commands shown in Figures 4-2 and 4-3, plus some of the line-drawing characters from the IBM extended character set, to print something you could use to support your proposal.

Printing this example also lets you see how closely your printer follows the Epson-compatible and HP-compatible commands and IBM extended character set.

❑ **NOTE:** *Not all Epson and Epson-compatible printers use the IBM extended character set. Some printers can be switched from the Epson mode to the IBM mode. (See "If your printer isn't compatible" later in this chapter.) Switching modes might cause the loss of other characters and character formats, such as italics and elite pitch.*

You might also need to switch font cartridges on an HP or HP-compatible laser printer to have access to the IBM extended character set. The internal fonts and the type of font cartridge determine which fonts, typestyles, and pitches are available.

Use a text editor, a word processor, or Edit or Edlin to create a file named PSAMP.DOC and enter the lines shown in Figure 4-7. If you have an HP-compatible printer, create the file PCLSAMP.DOC, as shown in Figure 4-8.

Some tips on entering the lines:

- The periods in lines 4–8 of both Figures 4-7 and 4-8 stand for spaces. Press the Spacebar once for each period.

```
 1:  ⌂@
 2:  Sample chart:
 3:  <Alt-201> type <Alt-205> 20 times <Alt-187>
 4:  <Alt-186>.......⌂GWe.Can⌂H.......<Alt-186>
 5:  <Alt-204> type <Alt-205> 20 times <Alt-185>
 6:  <Alt-186>.<Alt-254>.Improve.⌂4quality⌂5..<Alt-186>
 7:  <Alt-186>.<Alt-254>.Increase.⌂4revenue⌂5.<Alt-186>
 8:  <Alt-186>.<Alt-254>.Decrease.⌂4costs⌂5...<Alt-186>
 9:  <Alt-200> type <Alt-205> 20 times <Alt-188>
10:
11:  These charts ⌂-1take less time⌂-0
12:  These charts ⌂-1cost less⌂-0
13:  And we can make these charts ⌂-1⌂4ourselves⌂-0⌂5
```

Figure 4-7. *PSAMP.DOC: contents of a sample chart file for Epson printers.*

```
 1:  ⌂E⌂(10U
 2:  Sample chart:
 3:  <Alt-201> type <Alt-205> 20 times <Alt-187>
 4:  <Alt-186>.......⌂(s3BWe.Can⌂(s-3B.......<Alt-186>
 5:  <Alt-204> type <Alt-205> 20 times <Alt-185>
 6:  <Alt-186>.<Alt-254>.Improve.⌂(s1Squality⌂(s0S..<Alt-186>
 7:  <Alt-186>.<Alt-254>.Increase.⌂(s1Srevenue⌂(s0S.<Alt-186>
 8:  <Alt-186>.<Alt-254>.Decrease.⌂(s1Scosts⌂(s0S...<Alt-186>
 9:  <Alt-200> type <Alt-205> 20 times <Alt-188>
10:
11:  These charts ⌂&d0Dtake less time⌂&d@
12:  These charts ⌂&d0Dcost less⌂&d@
13:  And we can make these charts ⌂&d0D⌂(s1Sourselves⌂&d@⌂(s0S
14:  <Alt-12>
```

Figure 4-8. *PCLSAMP.DOC: contents of a sample chart file for HP printers.*

- In lines 3, 5, and 9, don't type the italicized words; instead, follow the instructions they give—enter the character whose code is 205 (shown as Alt-205) 20 times. You may want to use the Cut and Paste or the Copy command of your word processor or text editor to avoid typing Alt-205 60 times in three lines.
- In line 6 there are two spaces before the final Alt-186, in line 7 there is one space, and in line 8 there are three spaces.
- The right edge of the box that appears when you type lines 4, 6, 7, and 8 is four columns to the right of the other lines because the printer commands in those lines take up space on the display. When you print the file, however, the commands will be carried out, not printed, and the edge of the box will be smooth.

Figure 4-8 shows the same file using HP PCL commands similar to the Epson commands in Figure 4-7.

After you have entered the lines into PSAMP.DOC or PCLSAMP.DOC, proofread the file against the appropriate Figure and save it. Exit to DOS, and print the file by typing the following Copy command:

```
C>copy psamp.doc prn
```

or

```
C>copy pclsamp.doc prn
```

Your sample chart should look like Figure 4-9.

```
Sample chart:

┌──────────────────────┐
│       We can         │
├──────────────────────┤
│ ▪ Improve quality    │
│ ▪ Increase revenue   │
│ ▪ Decrease costs     │
└──────────────────────┘

These charts take less time
These charts cost less
And we can make these charts ourselves
```

Figure 4-9. *Sample chart.*

If your printer output looks like Figure 4-9, your printer closely follows the Epson-compatible or HP-compatible commands and the IBM extended character set. Even if your word-processing program prints different fonts and lets you use the IBM extended characters, you can still make good use of knowing how to control the printer yourself. If you have an Epson-compatible or HP-compatible printer that supports the IBM extended character set, skip the next section and go on to the heading "Changing the Sample Chart with One Command."

If your printer isn't compatible

If your printer doesn't follow the IBM extended character set, your output may look something like Figure 4-10. (Some printers may even print the

Framingham State College
Framingham, Massachusetts

```
Sample chart:
IMMMMMMMMMMMMMMMMMM;
:          We can       :
LMMMMMMMMMMMMMMMMMM9
: ^ Improve quality     :
: ^ Increase revenue    :
: ^ Decrease costs      :
HMMMMMMMMMMMMMMMMMM<

These charts take less time
These charts cost less
And we can make these charts ourselves
```

Figure 4-10. *Sample chart from an incompatible printer.*

"graphics characters" in italics.) If your printer doesn't follow the Epson-compatible or HP-compatible commands, the boldface and italic words in Figure 4-9 may appear in normal print, or the output may differ in some other way.

Each character used to draw the box (I ; L 9 H < M and :) has a code that is 128 less than the character you entered (compare Figure 4-10 to Figure 4-9). Figure 4-11 shows the relationship of these characters to the characters you entered. This is how some printers handle characters they can't print.

That extra printer command, ⎋*(10U*, in line 1 of Figure 4-8, is designed to switch on the *PC-8 character set* in the LaserJet and avoid this problem with the special characters. The PC-8 character set contains all the IBM extended ASCII codes, so the output looks just like Figure 4-9.

Other printers may define a different extended character set for codes 128–255. If this is the case, your printer's output won't look like either

Character Entered	***Code***	***Less 128***	***Character Printed***
╔	201	73	I
╗	187	59	;
╠	204	76	L
═	205	77	M
║	186	58	:
╣	185	57	9
╚	200	72	H
╝	188	60	<
■	254	126	~

Figure 4-11. *Effect of subtracting 128 from the IBM extended character codes.*

Figure 4-9 or Figure 4-10. You may, however, be able to use extended character codes. Look in your printer's manual for the character-set chart (probably in an appendix) and follow that chart instead of Appendix C.

Your printer may be able to print the IBM extended character set if a switch is set correctly. Check your printer's manual to see if the switch setting is correct—the switch may be called a *DIP switch*, because it's one of four or eight tiny switches called a *Dual Inline Package*. For example, the popular Panasonic dot-matrix printers have a switch that allows them to print the IBM extended character set. Even some Epson printers use DIP switches to switch between Epson mode and IBM mode. When you do this with the Epson FX-85, for example, the printer will print the IBM extended character set but will no longer print italic characters. If you need to, change the switch setting, turn the printer off and back on, and copy PSAMP.DOC to the printer again.

Similarly, even if your printer doesn't use the same commands as shown here, you can still control the printer. Again, you'll have to go to your printer's manual and find a summary of the printer commands (probably in another appendix). You most likely can use the preceding examples; just substitute your printer's commands for the ones shown.

If your printer doesn't follow the commands shown here, and you haven't found the commands in your printer's manual, skip the following two sections and go to the heading "Experimenting with Your Printer."

Changing the Sample Chart with One Command

Modern computer printers are remarkably versatile. Once you have entered text into a file, you can quickly change its appearance. The following changes, for example, require adding or changing just one printer command, yet each changes the appearance of the entire file output. Taking a few minutes to try these alternate versions of PSAMP.DOC or PCLSAMP.DOC (the file you created earlier) should give you a better feeling for what your printer can do. Load PSAMP.DOC or PCLSAMP.DOC with your editing program, make the described change, save the revised version, exit to DOS, and copy PSAMP.DOC or PCLSAMP.DOC to the printer:

- For Epson printers: Insert a line that contains ⎋*E* after the first line in the file. This prints the file in boldface. For HP printers: Insert the command ⎋*(s4101T* at the end of the first line. This prints the file in the Times font (if you have it).
- For Epson printers: Change the ⎋*E* you just inserted to ⎋*M*. This prints the file in Elite type (a smaller type—12 characters per inch—than the

standard 10 characters per inch). For HP printers: Put the command ⎋*(&l10* (L-one-zero) at the end of the first line. This prints the file in landscape mode.

- For Epson printers: Change the ⎋*M* in the second line to character 15. (Type either Alt-15 or Ctrl-O, whichever your editing program accepts.) This prints the file in compressed print (17 characters per inch). For HP Printers: Add the command ⎋*(s14V* to the end of the first line. This scales the font printed to 14 points, providing the current font is scalable.

You may want to try changing a text file in other ways by using the summary of printer commands in Appendix B and the character set charts in Appendix C. The results on your HP LaserJet printer will vary, depending on mode, installed cartridges, and soft fonts. Refer to the font manual that comes with your printer, or order the printer's technical reference manual for additional information.

Be Judicious with the Reset Command

The Printer Reset command—⎋@ for Epson printers and ⎋E for HP—cancels all outstanding printer commands and returns the printer to the settings that were in effect when you first turned it on. These settings include normal (or draft-quality) print, 6 lines per inch, and 66 lines per page. But the Reset command also resets such characteristics as the page-depth counter (by which the printer keeps track of how many lines it has printed), the left margin, and even the printer's buffer (where the printer stores the data to be printed). Using Printer Reset can cause unexpected results; if you put a Reset command at the end of a line, for example, the line probably won't be printed because the Reset command clears the line from the printer buffer before the printer can print the line. On a LaserJet, the Reset command will eject a page in the middle if used inappropriately.

Because of the possible side effects, you should put a Printer Reset command only at the beginning of a file, where you want to guarantee the standard settings. It's best to avoid putting a Reset command in a file that will be printed with a word processor. The word-processing program itself keeps track of the printer's state, and resetting all the conditions to the standard settings can cause strange results, such as the wrong type of print, irregular margins, or page breaks that occur in the wrong place.

Experimenting with Your Printer

This chapter covered only a few of the available printer commands. Chapter 12, which shows several interesting (and potentially useful) ways to use your printer, covers a few more commands, and Appendix B summarizes the common printer commands for Epson-compatible and HP-compatible printers. But there are still more—dozens in all; some printers have more than 100 commands. There isn't room enough in this book to describe all the printer commands and still cover the other topics.

Besides installation instructions and summary tables, many printer manuals contain excellent descriptions of each command, complete with an example and sample output. It's definitely worth a look—you may never need most of these commands, but spending some time putting commands in files and printing the results might give you several ideas for new ways to use your printer.

Chapter Summary

- Printer commands control print attributes such as type of print, line spacing, starting a new page, and graphics.
- Most printer commands start with the Escape character (code 27). Because you can't type the Escape character, you must put these commands in a file and print the file.
- You can enter printer commands that don't start with the Escape character by redirecting the output of an Echo command or by copying from the console to the printer.
- You can put commands in a file using any text editor or word processor that accepts non-keyboard characters.
- Use the Printer Reset command with care; it can cause unexpected results, especially if you put it in a file that will be printed with a word-processing program.
- Chapter 12 shows some ways to make your printer more versatile and describes a few more printer commands.
- Appendix B summarizes the most frequently used printer commands for Epson-compatible and HP PCL-compatible printers.

CHAPTER

5

Debug: A Special Sort of Editor

Debug is a DOS program designed to help programmers find errors, or *bugs,* in their programs. It's a special-purpose editing program with some features especially suited to working with programs in assembly language. But Debug isn't limited to program files; you can use it to examine and change any file, no matter what it contains or what application program created it.

Debug doesn't look at a file as a collection of words, sentences, and paragraphs, but simply as a series of bytes. It cannot compose or format text, so it's not suitable for routine editing of text files. Nor does Debug provide the built-in convenience or protections of a word processor or text editor, so if you are working with valuable files, protect yourself by making a copy of the file first. Because Debug makes so few assumptions about what a file contains, however, sometimes it's just the tool you need.

This chapter shows how to use many of Debug's capabilities, some of which are used in other parts of this book.

Bytes, Hexadecimal, and ASCII

Debug lets you work with the individual bytes of a file, and it understands only hexadecimal numbers. If you haven't yet read Chapter 1, which describes a byte, hexadecimal numbers, and the ASCII code, you should take the time now to do so because Debug requires some understanding of all three. A brief review:

- A *byte* is the basic unit of computer memory and disk storage. It is an area large enough to hold a number from 0 through 255.
- *Hexadecimal* is the base-16 number system normally used to keep track of memory addresses and other computer-related numbers. It uses the same digits as the familiar decimal (base-10) system for the numbers 0 through 9 but uses the letters A through F to represent the decimal numbers 10 through 15. Remember, 10 in hexadecimal is 16 in decimal; to keep your number systems straight, try reading it as "one-zero" rather than "ten."
- *ASCII* is a code that computer manufacturers and users have agreed to use to represent certain functions, letters, numbers, and other symbols. Because 1 byte can hold a number from 0 through 255, it can represent any of 256 possible characters; the ASCII code specifies the characters for only the first 128 (codes 0 through 127), but the IBM PC and most IBM-compatible computers also define characters for the remaining 128 (codes 128 through 255). Appendix C shows the correlation between decimal numbers, hexadecimal numbers, and ASCII codes.

Working with Files

Like a word processor or text editor, Debug lets you examine a file, change it, and store the revised version. As you'll see, however, Debug displays the file quite differently; although many of its commands perform functions similar to those of other editing programs, they, too, have a different look and feel. But there are enough similarities that you can quickly learn to use some of Debug's important capabilities.

Most word processors let you move the cursor anywhere on the screen to make changes; for this reason, these programs are often called full-screen editors. Editor is a full-screen editor. Edlin, on the other hand, is a simple text editor. It doesn't let you use the cursor to move through the file, and it limits you to working on one line at a time. (Edlin is often called a line editor.) Although a line-editor program does much less than a word processor, it's usually smaller and faster than a word processor, and it's well suited for small jobs such as editing batch files.

Like a line editor, Debug doesn't let you use the cursor to move through the file. It doesn't even let you work on lines, in fact, because it doesn't recognize lines. Debug treats a file as simply a series of bytes; you work on the file a byte at a time. It would be painstaking to edit a text file with Debug, but its approach is quite useful when you're working with other types of files such as programs, printer commands, and data files for programs.

A quick tour of Debug will show you both its similarities and its differences compared to other editing programs. The simplest approach is to create a small text file and use Debug to examine and change it.

First create a text file by copying from the console to a file named TEST.DOC:

```
C>copy con test.doc
This is item 1.
This is item 2.
And this is the last item.
^Z
```

Now display the file with the Type command to confirm its contents:

```
C>type test.doc
This is item 1.
This is item 2.
And this is the last item.
C>
```

You Need the File DEBUG.EXE

Like other DOS external commands such as Diskcopy and Edlin, Debug's command file, DEBUG.EXE (or DEBUG.COM for older versions of DOS), must be in either the current directory or a directory you have named in a Path command. If DEBUG isn't in any directory, copy it to the appropriate directory on your working disk from the DOS disk that contains it.

Using Debug to Examine and Change a File

To start Debug, you enter its name, followed by the name of the file you want to examine or change. The test file is named TEST.DOC, so enter:

```
C>debug test.doc
-
```

Debug responds by displaying a hyphen followed by the cursor. The hyphen is Debug's command prompt, like the DOS system prompt (such as C>) or the asterisk displayed by Edlin. (Don't mistake this hyphen for a minus sign. You needn't type the hyphen; it appears automatically.) Debug is waiting for you to enter one of its commands.

Displaying a File with the Dump Command

To display the file, you type *d*, which is short for *Dump* (not display). Despite its slightly unsavory name, dump doesn't reflect on the usefulness of the command or its output; like other computer terms, dump acquired its own computer-related meaning in the dawn of computing, around 30 years ago. When a program went awry on one of those ancient, room-filling machines, sometimes the only way to find the problem was to print everything in the machine's memory, byte by byte, in hexadecimal, then search through hundreds of pages of these numbers. These mammoth printouts weren't formatted to make them easier for humans to read; they looked as if the computer had been tipped on its side and the data dumped on the floor. Hence *dump*.

Debug's Dump command displays the contents of the computer's memory byte by byte in two forms: the number contained in each byte in hexadecimal (it can vary from 0 through FF, which is 255 decimal), and the ASCII character that number represents. If you load a file into memory with Debug and type the Dump command, the display shows the contents of the file.

To display the memory where Debug loaded TEST.DOC, enter:

```
-d
```

Debug responds by displaying the first 128 bytes starting at the address where it loaded TEST.DOC, as shown in Figure 5-1.

```
xxxx:0100  54 68 69 73 20 69 73 20-69 74 65 6D 20 31 2E 0D  This is item 1..
xxxx:0110  0A 54 68 69 73 20 69 73-20 69 74 65 6D 20 32 2E  .This is item 2.
xxxx:0120  0D 0A 41 6E 64 20 74 68-69 73 20 69 73 20 74 68  ..And this is th
xxxx:0130  65 20 6C 61 73 74 20 69-74 65 6D 2E 0D 0A xx xx  e last item...xx
xxxx:0140  xx xx xx xx xx xx xx xx-xx xx xx xx xx xx xx xx  xxxxxxxxxxxxxxxx
xxxx:0150  xx xx xx xx xx xx xx xx-xx xx xx xx xx xx xx xx  xxxxxxxxxxxxxxxx
xxxx:0160  xx xx xx xx xx xx xx xx-xx xx xx xx xx xx xx xx  xxxxxxxxxxxxxxxx
xxxx:0170  xx xx xx xx xx xx xx xx-xx xx xx xx xx xx xx xx  xxxxxxxxxxxxxxxx
```

Figure 5-1. *Dump command output.*

This isn't as formidable as it looks, although the file certainly isn't as easy to read as when you display it with the DOS Type command. But the right side of the screen contains some reassuringly familiar words, and there is a certain regularity to the display.

You won't see all the little *x*'s on your display, but the contents of the last four lines of your display will differ from anyone else's. Those lines contain whatever happens to reside at the memory addresses that follow TEST.DOC on your computer. Rather than confuse everyone, throughout this chapter a lowercase *x* will be used to represent a part of the display that may vary. (It really isn't significant to the immediate topic, anyway.)

To best understand how the Dump command displays a file, it helps to break down the image on your screen into individual elements.

Each line of the Dump command's output shows 16 bytes of memory. Remember, the file TEST.DOC was loaded into memory from disk by Debug. So what you're looking at is computer memory, not the file on disk.

The number 16 is important here. The decimal number 16 is shown as 10 in hexadecimal (you'll see the significance of that in a moment). The file TEST.DOC is 62 bytes long, which is 2 bytes shy of 64, or 16 × 4. So the file takes up three full lines and only 14 bytes of the fourth line.

Each line is divided into three columns or groups: the hexadecimal values at the beginning of each line are memory *addresses;* in the middle are the hexadecimal *byte values* of the 16 bytes located at the address; and to the right are the *ASCII characters* that correspond to the byte values.

Figure 5-2 shows how the first line of the preceding display breaks down into those three groups.

```
Address                          Byte values                          ASCII characters

           0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
xxxx:0100  54 68 69 73 20 69 73 20-69 74 65 6D 20 31 2E 0D  This is item 1..
```

Figure 5-2. *The three groups of a Dump command output line.*

To determine a specific address, replace the last digit in the address (on the left) with the hexadecimal value shown over the middle group (above). The byte 6D, for example, is located at address 10B. You can say that address 10B contains the hexadecimal value 6D (109 decimal) and corresponds with the lowercase m in the ASCII character set.

The address

The left group (the first nine columns) contains the *address* of the first byte displayed in the line. The address consists of two parts. The first four digits (up to the colon) tell you in which *memory segment* the bytes are located. This broadly defines where the bytes are located in the computer's memory, much as an area code broadly defines the location of a telephone exchange. The memory segment number varies (which is why it is represented here by *x*'s), and its value depends on how much memory is installed in your computer and how much memory was available when you started the Debug program. For this brief look at Debug, you can ignore the memory segment numbers, just as you can ignore the area code if you're dialing a local telephone number.

The second part of the address, the last four digits, is referred to as the *offset*. This number contains the information that's most useful to you. The address of the first byte of the first line of the file is 100. The next line down shows a value of 110 hexadecimal. That's not 10 bytes greater but *16* bytes greater. (Remember, 10 hex is 16 decimal.)

The reason the offset is important is that it tells you how to locate specific bytes within the file. For example, the address of the second byte is 101, of the third byte is 102, and so on to the sixteenth and last byte, whose address is 10F. The next line starts again with address 110.

Going deeper, the address of the fourth byte in the fifth line of Figure 5-1 (which begins with *xxxx*:0140) is 143. (The fourth byte is 143? Remember: The first byte is 0.) The address of the last byte in the file is 13D. (The byte at that location is 0A hexadecimal, or 10 decimal.) As you can see in Figure 5-1, the address of the first byte of each line is 10 (hexadecimal) higher than the previous line. Hexadecimal 10 is 16 decimal, so each line contains 16 bytes.

The byte values

The middle group, separated from the address by two spaces, contains 16 pairs of numbers. These show the hexadecimal value of the 16 bytes that start at the address at the beginning of the line. For example, the address of the first byte of the second line is 110. That byte contains 0A hexadecimal (10 decimal); the address of the second byte is 111, and it contains 54 hexadecimal (84 decimal); and so on through the sixteenth and last byte of the second line, whose address is 11F, which contains 2E hexadecimal (46 decimal). The hyphen in the middle breaks the 16-byte lines into two groups of eight for easier reference.

The ASCII translations

The right group (the last 16 columns) is separated from the byte values by three spaces. It shows the ASCII equivalent of the corresponding byte values in the middle group. For example, the first character on the first line is T, which is the ASCII character for 54 hexadecimal (84 decimal); the second character is h, the ASCII character for 68 hexadecimal (104 decimal).

Debug shows only text characters, numbers, and punctuation marks in this group. If you examine the display, you'll see that only characters from the space, whose ASCII code is 20 hexadecimal (32 decimal), through the tilde (~), whose ASCII code is 7E hexadecimal (126 decimal), are shown. All other characters are shown as periods. For example, the carriage return, which is the last character in the first line (its value is 0D hexadecimal, or 13 decimal), is shown as a period; it's outside the range of the standard text characters.

The carriage-return and line-feed characters

TEST.DOC contains three lines. From the text you can see where each line begins, but you can also find the lines by looking for the hexadecimal values of two special ASCII characters: carriage return and line feed.

0D (13 decimal). This character is called *carriage return* in the ASCII character set, even though most of today's computers and computer terminals have nothing that resembles a carriage. The term is a carryover from the time when most of the computer terminals in use were Teletype terminals or other typewriterlike devices with carriages that had to be returned at the end of a line. The analogous operation on a display terminal is moving the cursor back to the beginning of the line.

0A (10 decimal). This character is called *line feed* in the ASCII character set. This term, too, is a carryover from the days of typewriterlike terminals; feeding paper into the machine rolled the paper up to print the next

line. The analogous operation on a display terminal is moving the cursor down to the next line.

DOS uses both characters, in the sequence 0D 0A, to mark the end of a line of text. The carriage-return character moves the cursor back to the beginning of the line, and the line-feed character moves the cursor down one line. This sequence of characters appears at addresses 10F–110, 120–121, and 13C–13D, the ends of the three lines.

Getting familiar with Dump command output

Once you understand how the Dump command displays a file, the rest of Debug comes pretty easily. Spend a bit of time now studying the Dump command output on your screen, referring to Figures 5-1 and 5-2, and comparing the ASCII characters on the right with the corresponding byte values in the middle group, until you can find the byte at a particular address or the address of a particular byte. There will be a quiz.

Quiz

Here are a few questions you can use to test your understanding of Debug's Dump command. Cover the column at the right and reveal it only when you're sure of the answers.

What is the value of the byte at address 119?	69
What character does it represent?	i
What is the address of the beginning of the word *last*?	132
What address is 6 bytes past the beginning of the word *item* in the first line?	10E
What is the value of that byte?	2E
What character does it represent?	Period (.)
What is the address of the next-to-last byte in the file?	13C
What is its value?	0D
What character does it represent?	Carriage return

Give yourself 10 points (hexadecimal) for each correct answer.

The Dump Command

The Dump command displays the contents of a portion of the computer's memory in the format just described. The Dump command also has two optional parameters that let you tell Debug where to start dumping memory and, if you wish, where to stop dumping memory:

d *start stop*

start is the hexadecimal address where Debug starts dumping memory.

stop is the hexadecimal address where Debug stops dumping memory. If

you omit *stop*, Debug dumps 128 bytes (eight rows of 16 bytes each) beginning with the *start* address.

If you omit both *start* and *stop* (enter just *d*), Debug starts dumping memory at the "current" address, or the memory location that follows that used by the most recent Dump command. This lets you page through memory 128 bytes at a time simply by entering *d*. If you don't tell Debug where to start dumping memory in the very first Dump command you use with a file, Debug starts at address 100.

For example, the Dump command you entered a while ago displayed the 128 bytes from 100 through 17F. Enter another Dump command with no parameters:

```
-d
```

Debug displays the 128 bytes of memory from 180 through 1FF. The output isn't shown here because the contents will vary from computer to computer.

To display just the area of memory from 122 through 130, enter the following Dump command:

```
-d 122 130
```

Debug displays only the bytes in the range you specified, from address 122 through address 130:

```
xxxx:0120       41 6E 64 20 74 68-69 73 20 69 73 20 74 68    And this is th
xxxx:0130  65                                                e
```

Another optional parameter you can use with Debug is *l* for length:

d *start l length*

length is a hexadecimal value telling Debug how many bytes to list following the address *start*. If *l* isn't specified, Dump displays only 128 bytes.

As an example, you know that the file TEST.DOC is 62 bytes long. That value in hexadecimal is 3E. To display only the contents of TEST.DOC, enter the following Dump command:

```
-d 100 l 3e
```

Debug will display 3E (62) bytes starting at memory location 100. Only the contents of TEST.DOC will be seen on your display.

Changing an Area of Memory

Although looking at a file is sometimes all you want to do—and Debug lets you see *exactly* what's in a file in a way that most word processors can't match—you often want to change the file, too. Debug lets you enter data into the computer's memory, replacing whatever was there before. If you load a file into memory, enter new data into that memory, and then write the memory back to the file, you have changed the file.

The Enter Command

The Enter command, *e*, lets you enter data into memory as byte values or as a string of characters. The Enter command has two parameters, the address where the data is to be entered and the data itself:

e *address data*

address is the address of the bytes in memory you want to change. If you enter just 1 byte, the byte at *address* is changed. If you enter more than 1 byte, the changes begin at *address*.

data is the data to be entered in memory. To change 1 byte, type the two-digit hexadecimal value to be entered or type a character in quotation marks (a one-character string). To change more than 1 byte, you would type a series of two-digit hexadecimal numbers with a space after each, or a multicharacter string (two or more characters that are enclosed in quotation marks).

Changing a series of bytes

To change the first line of TEST.DOC, enter:

```
-e 100 "Changed by Debug."
```

To verify the change, display memory again with the Dump command. TEST.DOC occupies the 62 bytes of memory from 100 through 13D. Dump that memory by entering:

```
-d 100 13d
xxxx:0100  43 68 61 6E 67 65 64 20-62 79 20 44 65 62 75 67  Changed by Debug
xxxx:0110  2E 54 68 69 73 20 69 73-20 69 74 65 6D 20 32 2E  .This is item 2.
xxxx:0120  0D 0A 41 6E 64 20 74 68-69 73 20 69 73 20 74 68  ..And this is th
xxxx:0130  65 20 6C 61 73 74 20 69-74 65 6D 2E 0D 0A       e last item...
```

The memory into which Debug loaded TEST.DOC doesn't start with *This is item 1.* anymore; it starts with *Changed by Debug*. Equally important (as

you'll see in a moment), the carriage-return and line-feed characters (0D and 0A) at 10F and 110 have been replaced. The rest of the memory occupied by the file remains unchanged.

Instead of specifying the data to be entered (*Changed by Debug.*) as a string, you could have made the same change by entering:

```
e 100 43 68 61 6E 67 65 64 20 62 79 20 44 65 62 75 67 2E
```

Obviously, the string method is easier when you're entering data that can be typed from the keyboard. When you're changing nontext data, however, you may find it quicker to change a series of bytes by entering the two-digit hexadecimal numbers. Note that you can combine string characters and hexadecimal numbers in one Enter command.

Changing a single byte

The byte at address 11E has the value 32 hexadecimal (50 decimal), the ASCII code for the character 2. The ASCII code for B is 42 hexadecimal (66 decimal), so change the 2 to a B by entering:

```
-e 11e 42
```

Again, verify the change by dumping the same area of memory you dumped earlier.

```
-d 100 13d
5187:0100  43 68 61 6E 67 65 64 20-62 79 20 44 65 62 75 67  Changed by Debug
5187:0110  2E 54 68 69 73 20 69 73-20 69 74 65 6D 20 42 2E  .This is item B.
5187:0120  0D 0A 41 6E 64 20 74 68-69 73 20 69 73 20 74 68  ..And this is th
5187:0130  65 20 6C 61 73 74 20 69-74 65 6D 2E 0D 0A        e last item...
```

Now the byte at 11E has the value 42 instead of 32 (both hexadecimal).

Storing a file with a different name

Although you changed the memory where Debug loaded TEST.DOC, the file itself remains unchanged. After all, you haven't written anything to disk.

You can write any area of memory to any file using Debug. To keep track of the different versions of the file, you'll save each version under a different name. This requires two steps: changing the file name with the Name command, and then writing the area of memory to disk with the Write command.

The Name command

The Name command changes the name Debug will use the next time you load a file into memory or write an area of memory to a file. It has only one parameter, the file name:

n *filename*

filename is the file name that Debug will use in the next Load command, which loads a file into memory, or in the next Write command, which writes an area of memory to a file.

For example, enter the following to change the name Debug will use the next time you write a file to TEST1.DOC:

```
-n test1.doc
```

There is no apparent effect, and there will be no real effect until you enter a Load or Write command.

The Write command

The Write command writes an area of memory to the file that either was loaded by the Debug command or was most recently named with the Name command. It has one parameter, the starting address of the area of memory it is to write to a disk file:

w *start*

start is the address of the beginning of the area of memory that Debug is to write to disk. If you omit *start*, Debug starts at address 100.

To write from memory to disk starting at address 100, enter:

```
-w
```

Debug responds by telling you (in hexadecimal, of course) how many bytes it wrote:

```
Writing 003E bytes
```

3E in hexadecimal is 62 in decimal (you can ignore the leading zeros). Debug wrote 62 bytes from memory to the file you named TEST1.DOC. Because you didn't tell the Write command where to start, it started writing from address 100. (Nor did you tell Debug the file length, but it knew anyway. You'll find out how to change the file length in a later section.)

The TEST1.DOC file now contains the modified version of TEST.DOC you created with the two Enter commands.

The Quit command

The Quit command returns you to DOS. It doesn't save a file or perform any other processing, so make sure you have saved any work with the Write command before entering it. The Quit command has no parameters:

```
q
```

Enter the Quit command to return to DOS:

```
-q
```

Verifying Your Debug Work

To make sure you've got both files—TEST.DOC and TEST1.DOC—enter the following Directory command using the * wildcard character:

```
C>dir test*
 Volume in drive C is FORSYTHWOLF
 Volume Serial Number is 1810-1496
 Directory of C:\DOS\DEBUG
TEST     DOC      62   10-16-92  12:09p
TEST1    DOC      62   10-16-92   1:31p
        2 file(s)      124 bytes
                    2678784 bytes free
```

If TEST1.DOC isn't there, go back to the heading "Using Debug to Examine and Change a File" and start over again, making sure that you change the name with the Name command and save the area of memory with the Write command.

Now, to make sure the files are different, display both versions with two Type commands:

```
C>type test.doc
This is item 1.
This is item 2.
And this is the last item.

C>type test1.doc
Changed by Debug.This is item B.
And this is the last item.
```

In TEST1.DOC, you changed the first line to read *Changed by Debug*. The first two lines of TEST.DOC are combined in TEST1.DOC because the last two characters you entered with the Enter command (g.) replaced the carriage-return and line-feed characters that separated the first and second lines. DOS doesn't find a carriage return and line feed until the end of the former second line (which now reads *This is item B.* because you replaced the 2 with a B).

Adding to a File

The changes to TEST.DOC left its length unchanged. Changing the length of a file requires one more step, to make sure that Debug writes the correct number of bytes to disk. Start by loading TEST.DOC with Debug again:

```
C>debug test.doc
```

Debug loads TEST.DOC into memory starting at address 100. The file is 62 (3E hexadecimal) bytes long, so it fills memory from address 100 through address 13D. Dump this memory by entering:

```
-d 100 13d
xxxx:0100  54 68 69 73 20 69 73 20-69 74 65 6D 20 31 2E 0D  This is item 1..
xxxx:0110  0A 54 68 69 73 20 69 73-20 69 74 65 6D 20 32 2E  .This is item 2.
xxxx:0120  0D 0A 41 6E 64 20 74 68-69 73 20 69 73 20 74 68  ..And this is th
xxxx:0130  65 20 6C 61 73 74 20 69-74 65 6D 2E 0D 0A        e last item...
```

To add something to the end of the file, simply use an Enter command to change memory starting at the first byte after the end of the file. The last byte of the file is at 13D, so the starting address for the Enter command would be 13E. To add *** Postscript *** to the end of TEST.DOC, enter:

```
-e 13e "*** Postscript ***"
```

This adds 18 bytes, making a total of 80 bytes (50 hexadecimal) to be written to the file (from 100 through 14F).

Dump 50 hexadecimal bytes of memory starting at 100 by entering:

```
-d  100 14F
xxxx:0100  54 68 69 73 20 69 73 20-69 74 65 6D 20 31 2E 0D  This is item 1..
xxxx:0110  0A 54 68 69 73 20 69 73-20 69 74 65 6D 20 32 2E  .This is item 2.
xxxx:0120  0D 0A 41 6E 64 20 74 68-69 73 20 69 73 20 74 68  ..And this is th
xxxx:0130  65 20 6C 61 73 74 20 69-74 65 6D 2E 0D 0A 2A 2A  e last item...**
xxxx:0140  2A 20 50 6F 73 74 73 63-72 69 70 74 20 2A 2A 2A  * Postscript ***
```

That's what you want in the new file. But if you type a Write command now, Debug will write only 3E hexadecimal (62 decimal) bytes, just as it did before. Debug has no way of knowing that you now want to write 50 hexadecimal (80 decimal) bytes.

Just as you can tell Debug what file name to use, you can tell it how many bytes to write. It's not that complex, but it does take you just a bit deeper into the inner workings of the computer's microprocessor.

The Microprocessor's Registers

The microprocessor in your computer has several bytes of its own memory, divided into 14 areas called *registers*. Just as you use a check register to keep track of your checks, the microprocessor uses its registers to keep track of what it's doing. The only reason you need to care about registers—unless you plan to write programs—is that Debug uses one of these registers to control how many bytes it writes whenever you use the Write command.

The Register Command

The Register command lets you display or change the contents of the microprocessor's 14 registers. To display the registers, enter an *r*:

```
-r
```

Debug responds by displaying the names and contents of the microprocessor's registers, as shown in Figure 5-3. As usual, numbers that may vary (and don't matter) are shown as *xxxx*.

Each register is 2 bytes long. The Register command displays the two-character name of each register (AX, BX, and so on through IP), followed by an equal sign and the value stored in the register. Happily, you can ignore most of this.

The highlighted register in Figure 5-3, named CX, is the only one you need to care about. That's the one register that Debug uses to control how many bytes it writes when you enter a Write command.

(If you're looking for the fourteenth register, remember that the eight two-character abbreviations at the end of the second row represent contents

```
AX=xxxx  BX=xxxx  CX=003E  DX=xxxx  SP=xxxx  BP=xxxx  SI=xxxx  DI=xxxx
DS=xxxx  ES=xxxx  SS=xxxx  CS=xxxx  IP=xxxx   NV UP EI PL NZ NA PO NC
xxxx:0100 xx             xxxxxx
```

Figure 5-3. *Register command display.*

of the last register. Go ahead and ignore them as well. You can also ignore the third line of the output, which starts with an address. Only hard-core assembly language programmers get excited about that stuff.)

To change the contents of a register, enter the Register command with one parameter:

r *name*

name is the name of the register whose value you want to change. Debug prompts you for the new value. If you omit *name*, Debug displays the value of all registers, as shown in Figure 5-3.

The reason for this aside about registers is that you increased the number of bytes to be written to a file from 62 (3F hexadecimal) to 80 (50 hexadecimal). Debug looks at the CX register to see how many bytes to write to a file, so you must put the correct value (50) in CX. And in hexadecimal, of course.

First, enter a Register command, naming CX:

```
-r cx
```

Debug repeats the name of the register (in uppercase) and its current value (in hexadecimal), then prompts for a new value with a colon on the next line:

```
CX 003E
:_
```

Sure enough, CX contains 3E hexadecimal (62 decimal), which is the length of TEST.DOC. When you load a file into Debug, it reads the length of the file from the directory entry and puts the length into CX. That's how Debug knew how many bytes to write earlier when you changed memory and wrote TEST1.DOC to disk: You didn't change the length and you didn't change CX, so it wrote the same number of bytes that it read.

If you pressed the Enter key now without typing anything, the contents of CX wouldn't change. To change the value in CX, type the new value. You increased the total number of bytes to 50 hexadecimal (80 decimal), so after the colon, enter *50*:

```
-r cx
CX 003E
:50
```

Display the registers to verify that CX contains the correct value:

```
-r
AX=xxxx  BX=xxxx  CX=0050  DX=xxxx  SP=xxxx  BP=xxxx  SI=xxxx  DI=xxxx
DS=xxxx  ES=xxxx  SS=xxxx  CS=xxxx  IP=xxxx   NV UP EI PL NZ NA PO NC
xxxx:0100 xx             xxxxxx
```

Now Debug knows how many bytes to write. The next time you enter a Write command, Debug will write 50 hexadecimal (80 decimal) bytes. To keep all these versions separate, change the file name to TEST2.DOC with a Name command before entering the Write command. Enter:

```
-n test2.doc
-w
Writing 0050 bytes
```

Use the Quit command to return to DOS so you can verify the contents of the files:

```
-q
```

You should have three test files; check by using a DOS Directory command:

```
C>dir test?

 Volume in drive C is FORSYTHWOLF
 Volume Serial Number is 1810-1496
 Directory of C:\DOS\DEBUG

TEST     DOC      62   10-16-92  12:09p
TEST1    DOC      62   10-16-92   1:31p
TEST2    DOC      80   10-16-92   2:15p
        3 file(s)      204 bytes
                    2678272 bytes free
```

The files are there and the lengths are right. Again, you can make sure by displaying them. This time, however, instead of using three Type commands, use one Copy command:

```
C>copy test?.doc con
TEST.DOC
This is item 1.
This is item 2.
And this is the last item.
TEST1.DOC
Changed by Debug.This is item B.
And this is the last item.
```

```
TEST2.DOC
This is item 1.
This is item 2.
And this is the last item.
*** Postscript ***        1 file(s) copied
```

The *files(s) copied* message is on the same line as *** Postscript *** because you didn't enter a carriage return and line feed at the end of TEST2.DOC.

Why Use Such an Editor?

The clearest message from these examples is that it would be pretty painful to use Debug as a text editor. But you can create and modify files with Debug and, more to the point, Debug can handle any type of data. You'll use a word processor or text editor to create and change text files, but Debug is an ideal tool for working with files that contain nontext data, such as programs or commands that control the printer or display.

Creating a Small Program

Later in this book, you'll be introduced to a few small programs that work with DOS to make your computer more useful. To create these programs, you'll be using Debug—even if you don't know how to program. To make the process safe and painless, you won't use the Debug commands directly. Instead, you'll use your word processor or text editor to create a text file that contains Debug commands. Then you'll use input redirection to give the commands to Debug. Although this may seem like taking extra steps, this method is easy and popular. (Most magazines do the same thing for the same reasons.)

A Debug Script File That Creates a Program

Because a text file that contains a series of commands and other program data is analogous to a script for a play, such a file is often called a *script file* or simply a *script*. The remainder of this chapter shows you how to make a script file that instructs Debug to create a short program and write the program to a file. After creating the script file, you'll start Debug with its input redirected from the script file, then test the program that Debug produces.

The script file contains Debug commands that:

1. Create a short assembly language program

2. Set the file name to FLASH.COM

3. Set the CX register to the number of bytes in the program

4. Write the program to a file named FLASH.COM

5. Quit Debug and return to DOS

The last step is crucial. If you redirect a program's input from a file, the program *never* looks for input from the keyboard. If you don't put the Quit command in the script file, Debug keeps looking for input after the end of the file. You won't be able to stop Debug because it pays no attention to the keyboard. Your computer will run amok, and you'll have to restart DOS to regain control.

Figure 5-4 shows the script file that creates the program FLASH.COM. Enter those lines (Debug commands) into a text editor (such as Editor) or your word processor if it lets you store a file with no formatting codes. You could use the Copy command to copy from the console to the file, but you'll feel less pressure if you know that you can go back and correct any typing mistakes.

The line numbers followed by colons are for reference only; do not enter them into your script. The finished file should contain exactly what is shown following the colon on each line. Some commands should look familiar, such as the Register command in lines 13 and 14, Name in line 15, Write in line 16, and Quit in line 17. Everything else in this text file will be explained later in this chapter.

```
1:  a 100
2:  mov bh,0
3:  mov cx,7d0
4:  mov ah,2
5:  mov dx,0
6:  int 10
7:  mov ah,8
8:  int 21
9:  mov ah,a
10: int 10
11: jmp 105
12:
13: r cx
14: 16
15: n flash.com
16: w
17: q
```

Figure 5-4. *FLASH.SCR: Debug script file with line numbers.*

When you have entered the file, proofread it against Figure 5-4, correct any errors, and save it with the name FLASH.SCR. To make sure FLASH.SCR is on the disk, enter the following Directory command:

```
C>dir flash.scr

 Volume in drive C is FORSYTHWOLF
 Volume Serial Number is 1810-1496
 Directory of C:\DOS\DEBUG

FLASH    SCR      133   10-16-92   2:45p
       1 file(s)      133 bytes
                  2677760 bytes free
```

(Your file size may be different, depending on which editor you use.)

If DOS responds *File not found*, create the file again and be sure to save it as FLASH.SCR. FLASH.SCR fits on one screen, so display it with the Type command:

```
C>type flash.scr
```

Figure 5-5 shows the file. Proofread this against your output from the Type command. Make certain that the first line is a 100, that there is a blank line between jmp 105 and r cx (line 12 in Figure 5-4), and that the last line is q (a Quit command). If you omit or misplace these lines, Debug won't create FLASH.COM, and you'll have to restart DOS.

```
a 100
mov  bh,0
mov  cx,7d0
mov  ah,2
mov  dx,0
int  10
mov  ah,8
int  21
mov  ah,a
int  10
jmp  105

r cx
16
n flash.com
w
q
```

Figure 5-5. *FLASH.SCR: Debug script file without line numbers.*

Redirecting Debug's Input from a Script File

To redirect the input of Debug to a script file, you use the < redirection character. To start Debug and tell it to read its commands from FLASH.SCR instead of the keyboard, enter:

```
C>debug < flash.scr
```

If all goes well, your screen should display the commands in FLASH.SCR and Debug's responses. Figure 5-6 shows how your program should respond; again, *xxxx* represents those numbers that vary from system to system.

```
-a 100
xxxx:0100  mov  bh,0
xxxx:0102  mov  cx,7d0
xxxx:0105  mov  ah,2
xxxx:0107  mov  dx,0
xxxx:010A  int  10
xxxx:010C  mov  ah,8
xxxx:010E  int  21
xxxx:0110  mov  ah,a
xxxx:0112  int  10
xxxx:0114  jmp  105
xxxx:0116
-r cx
CX 0000
:16
-n flash.com
-w
Writing 0016 bytes
-q

C>_
```

Figure 5-6. *Debug's response to FLASH.SCR.*

Did it work?

If your system responds for a while and then stops (perhaps displaying what looks like several error messages), and you can't get any response from the keyboard, don't worry. You haven't damaged the machine or any files; there's probably a mistake in FLASH.SCR, and Debug is still looking for input from the file instead of from the keyboard. Do the following:

1. Turn the power switch off, wait 15 seconds or so, and turn the power switch back on. DOS will go through its startup routine.

2. Load FLASH.SCR with your word processor or text editor. Proofread it against Figure 5-5 until you're satisfied that it's identical. (Make sure that the first line is a 100, that there's a blank line at the end of the program statements [line 12], and that the last line is q, as described earlier).

3. Enter the Debug command again, redirecting input to FLASH.SCR. The responses to the Register and Write commands at the end of Figure 5-6 are just like the responses you saw when you used those commands for working with text files, earlier in this chapter. Debug treats commands in a script file just as if you had typed them.

The Name and Write commands in FLASH.SCR tell Debug to save 16 hexadecimal (22 decimal) bytes in a file named FLASH.COM. Make sure it was created by entering:

```
C>dir flash.com

 Volume in drive C is FORSYTHWOLF
 Volume Serial Number is 1810-1496
 Directory of C:\DOS\DEBUG

FLASH    COM        22   10-16-92   2:55p
        1 file(s)         22 bytes
                    2677248 bytes free
```

If FLASH.COM wasn't created, or if it isn't 22 bytes long, go back and proofread FLASH.SCR, correct any errors, and enter the Debug command again.

Testing your creation

You've created a program, but you may have no idea what it does. It's fairly simple: FLASH.COM waits for you to type a character, fills the screen with that character and waits for you to type another character, fills the screen with that character, and so on until you stop the program by pressing Ctrl-Break (or Ctrl-C if you don't have a Break key).

Try it. Because FLASH.COM is a command file (its extension is COM), all you need to do to run it is to enter its name:

```
C>flash
```

Nothing happens immediately. Now press the F key.

If the screen fills with the character you typed, type another character. You will find that the machine displays a screenful of characters almost as

fast as you can type just one. The program you created isn't very big, but it sure is fast. This program demonstrates just how fast your computer is.

To end the program, press Ctrl-Break (or Ctrl-C). This leaves your screen filled with the last character you typed. To make the screen less confusing, you can press the Spacebar (filling the screen with spaces) before you press Ctrl-Break. If you stopped the program with the screen filled with some other character, simply clear the screen by entering the DOS clear screen (cls) command.

If nothing happened at all, there's probably a mistake in the program. Press Ctrl-Break (or Ctrl-C). If that doesn't restore the system prompt, try to restart DOS by pressing Ctrl-Alt-Del. If that doesn't work, turn the power switch off for 15 seconds, then turn it back on. When DOS displays the system prompt, use your text editor or word processor to proofread FLASH.SCR and correct any mistakes. This time, pay special attention to the program statements (lines 2 through 11 in Figure 5-4).

Creating a Program with the Assemble Command

You just produced the program named FLASH.COM by creating a file of Debug commands with a word processor or text editor, then starting Debug with its input redirected to that file. You can also create that same program directly in Debug by using the Assemble command.

The Assemble Command

The Assemble command translates statements in *assembly language*—a semireadable language of instructions, or *mnemonics,* for the microprocessor—into a more compact form meaningful to the microprocessor but essentially unreadable by humans. This language is often called *machine language*. The Assemble command has only one parameter, the address where you want the translation to begin:

a *address*

address is the location in memory at which Debug is to begin storing the translated assembly language mnemonics you will enter. If you omit *address* the first time you enter an Assemble command, Debug begins storing the instructions at address 100. If you omit *address* with subsequent uses of the Assemble command, Debug begins storing the instructions at the first byte following the last batch of statements you entered with an Assemble command.

When you enter an Assemble command, Debug prompts you for the first instruction by displaying the starting address. After you type an instruction and press Enter, Debug stores the translated statement and prompts you for the next statement by displaying the address where it will be stored.

The Unassemble Command

To prove that anything done can be undone, the Unassemble command translates from machine language back to assembly language mnemonics. Like the Assemble command, Unassemble has one parameter, the address where you want the reverse translation to begin:

u *address*

address is the location at which Debug is to begin translating the contents of memory to assembly language statements. If you omit *address* the first time you enter an Unassemble command, Debug begins translating memory at address 100. If you omit *address* with subsequent uses of the Unassemble command, Debug begins translating at the first byte following the last area of memory you used with the Unassemble command.

It's no miracle

The Unassemble command is useful primarily to assembly language programmers. They can list their programming instructions and read through them—just as you or I would read through baseball statistics in the sports section. But a certain madness is required to really appreciate unassembled computer instructions. Although Debug's Unassemble command will show you the inner workings of just about any program, it will never tell you how or why something happens the way it does.

Later in this book, and as you continue to work with DOS, you'll discover some method behind the madness of the Unassemble command. For now, keep in mind that Debug can be used to translate machine language instructions into assembly language mnemonics if the need arises.

Chapter Summary

- You can use Debug to create, examine, or change any sort of file by loading the file into memory, using Debug commands to modify the memory where the file was loaded, and then writing the modified memory back to the file.
- Debug displays and accepts addresses and byte values in hexadecimal numbers. Byte values can also be changed by using text strings.
- Debug provides none of the built-in convenience or protections of a word processor or text editor. If you use Debug to work with a valuable file (text, data, or a program), protect yourself by making a copy of the file and working with the copy.
- Although Debug is poorly suited for working with text files, its capabilities make it an excellent tool for working with files that contain a mixture of text and other data, such as files of commands that control a printer or the display.
- If you redirect Debug's input to a script file, be sure the script file includes all input, especially the Quit command.

CHAPTER

6

Advanced Batch-File Techniques

Batch files are your key to making the computer work the way you work. If you don't use batch files, the computer controls how you use it. Batch files put you in charge.

This chapter gives some guidelines for designing batch files, describes several techniques for writing batch files, and demonstrates most of the techniques with short examples. The techniques apply to any batch file; the remainder of the book describes even more batch files that use these techniques to tailor the appearance and behavior of DOS to your needs.

This chapter also describes the Call command, which lets you use a batch command in a batch file just as you would use any other command. As you'll see, this significantly increases the power of your batch files.

Some Guidelines for Designing Batch Files

Program presentation is an important part of writing software. The best programming *code* in the world just doesn't cut it if the interface is sloppy, the program is awkward to use, and it looks ugly. Purists might not consider creating a batch file to be on the same level as writing software, but you're creating something that others will use on the computer. A clear interface and attractive look are just as important for a batch file as they are for any other piece of software.

Whether the subject is machines, manuals, programs, or batch files, *easy to use* almost always means *harder to make*. It takes many more hours to write a good Windows program than an old-fashioned DOS utility. A batch file that uses a minimum number of commands to perform a task isn't always the best solution. That batch file probably gives the user no clue as to how it's used. In fact, a lack of important feedback may lead to the misuse of a batch file and may even damage some files. A batch file that's easy to use and won't damage files isn't that hard to write, but it takes time. You must think about how the batch program will be used, what could go wrong, and how you want the screen to look.

Here are some guidelines that can help you design and produce batch files that are easy and safe to use:

- Keep the screen easy to read and attractive while the batch program is running. (The next section describes several techniques you can use to do this.)
- Display brief, helpful, or reassuring messages, especially if the batch command performs different operations. Make sure that you—or whoever uses the batch command—will be reassured that everything's OK.

- Avoid costly mistakes. If a batch command has several parameters—or if entering it incorrectly could damage a file—have the batch command display brief instructions if it's typed with no parameters. You'll appreciate the help if you use the batch command only once in a while, and it also lets someone else use the batch command with little or no instruction from you.
- Anticipate errors that someone might make, both when typing the batch-file name and when responding to prompts or menu selections while the batch-file program is running. Try to make it impossible for the person using the program to make an error (this technique is sometimes called *bulletproofing*).
- Provide conveniences. If the batch file creates temporary files, for example, erase them after they've served their purpose. Also, if someone might want to print a file after using the batch command, offer that as an option at the end of the batch file.
- Above all, *think*. After you think you're done writing the batch-file program, get up and walk around. Then sit back at the computer and run your program as if it's the first time you've ever seen it. Never assume anything. Then see if your program behaves the way you'd expect. (Sometimes this requires the enlistment of a friend or colleague; programmers get too involved with their "babies.")

Make Your Screens Attractive and Readable

Simple, uncluttered screens that give unambiguous instructions let anyone use a batch command quickly and confidently. The techniques for designing good screens are straightforward:

- Keep the screen free of distracting messages. Start the batch file with *@echo off*, and redirect Copy command output to NUL, the null output device. (You'll learn about that in a few sections.)
- Separate groups of related information with blank lines. Use Echo commands in batch files and blank lines in any files that the batch file displays using the Type command.
- Use ANSI display attributes such as reverse video, high intensity, and color (if your system uses it) to highlight important information. But don't overdo it: Remember the boy who cried "Wolf!"
- Use the extended graphics characters to frame related groups of information, to represent keys, or simply to make the screen more attractive. You can write batch files whose displays have the same professional appearance as most application programs.

There's only one "don't" when writing batch files:

- Don't have the batch file beep unless it's absolutely necessary. If overused, the beep not only loses its attention-getting quality, it quickly becomes an irritant. Save it for those few really important situations that deserve a shout.

Beeps and buzzes are necessary sometimes to get the attention of the person using the batch file. The DOS beep is OK, but it suffers from a lack of variety. Later in this chapter you'll be introduced to interesting utilities that let your computer do more than beep.

Conventions for Entering Batch Files

You'll notice that the batch files in this book follow a few conventions: Label names are in uppercase; commands are in lowercase; if a batch file includes labels, the commands are indented three spaces; and similar labels are used in different batch files when the circumstances are similar.

Conventions like these aren't for the benefit of the person who uses the batch file, but for the person who writes or revises the batch file. They don't affect what the batch command does; they just make it easier to see how the batch command works. You don't have to follow these conventions, but you might find that they make writing batch files easier and quicker, and that they really help if you decide to go back and change a batch file several months after writing it.

Some Hints for Writing Batch Files

Writing batch files involves some risks you usually don't face when you use an application program: You might do something that makes it necessary to restart DOS, and there's a chance you might inadvertently erase a file.

Don't panic!

There is no inherent danger to programming batch files, at least none that you don't encounter every day when you're using DOS. Fortunately, you can take a few steps to minimize any risks.

Before you start

The Break command controls how often DOS checks to see if you have pressed Ctrl-C (or Ctrl-Break). You want DOS to check as often as possible, so make sure Break is on when you're working on batch files. This is done by putting the following command in your AUTOEXEC.BAT file:

```
break=on
```

There is little penalty for setting Break on; some operations are a bit slower. So when you don't want it, just enter *break=off* at the system prompt.

Protect yourself against inadvertently changing or deleting valuable files by making them read-only with the Attribute command. Enter the following:

```
attrib +R filename
```

This is a good practice even if you're not working on batch files. For example, setting any file's status to read-only makes it hard to accidentally delete that file. Consider the following:

```
attrib +R \COMMAND.COM
```

Now it would be next to impossible to accidentally delete your system's COMMAND.COM file—a feat some beginners accomplish with amazing ease.

Work on your batch files in a separate directory, then copy the finished files to the \BATCH directory or wherever you keep your finished batch files. If you use a RAM disk when you work on batch files, be sure to copy from the RAM disk to a real disk fairly often; if you have to restart DOS, the contents of the RAM disk are lost. (For information about setting up and using a RAM disk, see Chapter 10.)

If a Batch File Doesn't Work Right

As you might know already, batch files don't always work the first time. (What does?) The bigger and more elegant the batch file, the more likely it is to behave perversely. Sometimes the problem is obvious, but there are times when finding and fixing the problem takes a combination of insight and patience. DOS gives you a few clues, and here's how to interpret them:

- If DOS displays *File not found* while a batch file is running, but the batch file doesn't use files, you might have used one or more of the redirection symbols (<, > or ¦) in an Echo command. You can't use <, >, or ¦ for anything except redirection—not even in a Remark command. DOS tries to carry out the redirection before it carries out the command. Instead, use the extended ASCII graphics characters, or put the redirection symbols into a file that is displayed by the Type command.

- If DOS displays *Bad command or file name* while a batch file is running, check the batch file for a misspelled command name. A CD command to the wrong directory can also cause that error; so can a stray pipe-redirection character (¦).
- If DOS displays *Intermediate file error* during a pipe command, make sure that there is a carriage return at the end of the last line of the batch file.
- If DOS displays *Label not found*, make sure that there is a colon before each label in the batch file.
- If you see *Syntax error*, then there is a problem with one of the batch-file commands. Usually, it's an If command missing an option on one side of the double equal signs—or you used only one equal sign instead of two. The For batch-file command can also produce the *Syntax error* message when you omit one of its parameters.
- If a batch file with labels doesn't carry out commands, repeats the same command series, or does nothing, press Ctrl-C and check the batch file for two identical labels. DOS searches for a label from the start of the batch file, so it never sees the second label.

If a batch file simply won't do what it's supposed to and gives you no clues about what's wrong, remove the initial *@echo off* command from the beginning and try the batch file again. With echo now *on*, DOS displays each command before it carries it out. This makes the screen a mess, but lets you follow along as the batch file runs. If lines on the screen scroll by too quickly, use Ctrl-S to stop and start the display. This usually leads you to the general area, if not the exact command, that is causing the problem. After you correct the problem, don't forget to replace the *@echo off* command at the beginning of the file.

And always remember that a batch file does exactly what you tell it to do. All you have to do is figure out exactly *what* you told it to do.

Use Separate Files for Large Menus or Displays

If a menu or other screen requires more than a few lines of text, it's usually quicker to put the text into a separate file and use a Type command to display it. The alternative is to use several lines of Echo commands, which typically display slowly and look awkward.

Displaying a text file with the Type command has some other advantages: It's usually easier to revise a text file than it is to change several Echo commands; the same text file can be used by more than one batch file; and you

can copy and revise the text file if another batch file requires a similar display.

The disadvantages of displaying text files are that each additional file takes up at least 2K bytes of disk space on a hard disk and that you must keep track of the text files as well as the batch file.

Preparing for the Examples

The rest of this chapter demonstrates some useful techniques for your batch files. The remaining chapters of the book describe several batch files, some of which you might want to keep or adapt for your own use. To keep your batch files together, create a subdirectory named BATCH in the root directory (or some place well suited for them) and change to that directory. You can do this by entering:

```
C>md \batch

C>cd \batch
```

To use these batch files from any other directory, add the \BATCH directory to your search path. Type a Path command with no parameters to display the current path. Then type another Path command that repeats the output of the first Path command, adding a semicolon and \batch at the end. For example, if your search path currently shows the C:\ and C:\DOS directories, the exchange would look like this:

```
C>path

PATH=C:\;C:\DOS

C>path=c:\;c:\dos;c:\batch

C>_
```

You might also want to add \BATCH to the Path command in your AUTOEXEC.BAT file so that your batch files are always available.

Some Useful Batch-File Techniques

You can apply the techniques described in the remainder of this chapter to any batch file, regardless of its purpose. In addition to the examples in this chapter, the techniques are used in batch files throughout the remaining chapters of the book.

Eliminating echo of batch commands

Turning echo off with an *echo off* command at the beginning of a batch file tells DOS not to echo the commands in the file, but DOS still echoes that first Echo command—announcing to all who run the program that it's a batch file. If you have DOS version 3.3 or later, you can prevent that by adding @ to the beginning of the echo command:

```
@echo off
```

DOS doesn't echo any batch command that begins with @. Indeed, you could leave out the initial *echo off* altogether and just start each line in the batch file with an @.

Echoing a blank line

Starting with DOS version 3.1, to echo a blank line, type *echo* immediately followed by a period:

```
echo.
```

If you type a space between the o and the period, echo displays the period. If you have a DOS version earlier than 3.1, you can echo a blank line by following the Echo command with a space and Alt-255 (hold down the Alt key, type *255* on the numeric keypad, and release the Alt key) which is a formfeed character.

Checking whether a parameter was typed

If you write a batch file that assumes the batch command will be entered with at least one parameter, be sure to consider what the effect would be if someone entered the command without a parameter. If you don't use the batch file frequently, you could forget the parameter, and someone else might not know how to use the batch file and not type the parameter at all. In particular, watch out for how you test the value of the parameters.

For example, suppose a batch file contains:

```
if not %1==fred goto END
```

What if the name of the batch file is entered and not followed by a parameter? DOS displays *Syntax error* and continues with the next command. The error message is distracting, perhaps even disconcerting. But, depending on what commands follow, further consequences can be much more serious because, even though the first parameter isn't fred, the batch command behaves as if it were.

To avoid this, enclose both the replaceable parameter symbol (% followed by a number) and the comparison string in quotation marks:

```
if not "%1"=="fred" goto END
```

Now DOS handles the absence of the parameter just like any other parameter value; it doesn't display an error message and processes the If command correctly, skipping any commands that follow the If command and going to the label END.

To check specifically for the absence of a parameter, use two quotation marks with nothing between as the comparison string:

```
if "%1"==""
```

You can also stick to just one similar item on either side of the double equal signs. The following is popular:

```
if !%1==!
```

Eliminating unnecessary DOS messages

Even though you suppress the echo of batch commands, some DOS commands also display messages that you might not want to see when you use them in a batch file. The Copy command, for example, displays a message telling you how many files it copied (such as *1 file(s) copied).* To suppress these messages, redirect the output of the DOS command to a device named NUL.

NUL isn't really a device like the printer (PRN) or console (CON); it's sort of a black hole, a device name that DOS lets you use when you want something to disappear. You can use the NUL device name in a command just as you would use any other device name. For example, enter the following command to copy the file named TEST.DOC in the root directory to the NUL device:

```
C>copy \test.doc nul
```

DOS responds *1 file(s) copied*, telling you that it copied the file, but there is no other copy of the file on the disk. The copy was sent to NUL, the black hole of DOS. (Play your choice of eerie music here.) The NUL device may seem pointless, but it does have its uses. Enter the Copy command again, but this time redirect the output of the command to the NUL device:

```
C>copy \test.doc nul > nul
```

Now nothing seems to happen. Both the copy of the file and the output of the Copy command—the message—were sent to NUL. If you name a real file or device as the destination of the Copy command (instead of the first nul above), but still redirect the output to NUL, DOS makes a copy of the file without displaying the confirming message; the silent approach.

So to eliminate messages produced by the Copy command in batch files, and any other DOS command that for some reason must display something, simply redirect the output of the Copy command to the NUL device.

Using the For command outside a batch file

You can type the DOS commands designed for use in batch files, such as Echo, If, Goto, and For, at the system prompt. Most of them, such as If and Goto, aren't especially useful outside a batch file. The For command, however, can be a quick way to carry out a single command on a series of files. For example, the Type command doesn't allow wildcard characters, but you could display all the files with the extension DOC by entering the following at the system prompt:

```
for %p in (*.doc) do type %p
```

If you typically type the same For command frequently at the system prompt, however, you're probably better off writing a short batch file.

Putting Echo to work

The Echo command can be used at the DOS prompt for a variety of interesting things. For example, if a page of paper needs to be ejected from the printer, and it's just too much for you to lean over and whack the Form Feed button, enter the following:

```
C>echo ^L > prn
```

That's the Control-L character, not the ^ character followed by an L. You can type this as Alt-12. As you remember from Chapter 1, Control-L is the ASCII control character that ejects a page from your printer, the formfeed character. (On some computers, typing a Ctrl-L also clears the screen.) And typing the above is also a handy way to eject a page of data that's "stuck" in your laser printer.

Making noise

Let's talk about that annoying beep. Just as you can echo a Ctrl-L, you can echo a Ctrl-G. Type the following:

```
C>echo ^G
```

There's no need to redirect output here; a Ctrl-G (or <*Alt*-7>) echoed to your display produces the desired audible effect. But can you imagine how annoying that would be in a batch file—especially if overused? Keep in mind that some DOS utilities do already make noise: Backup beeps when it wants you to insert a new disk, for example. But tossing an *echo* ^*G* into a batch file is a good attention getter if used sparingly.

The Sound command

There are utilities for creating noise on a PC. Not beautiful symphonic noise, but something a little less raw than the DOS ^G beep. One such utility, SOUND.EXE, is included on the disk offered with this book. This utility creates the Sound command, which beeps at a given frequency and for a given duration, allowing you to more liberally sprinkle sound into your batch files. The format for the Sound command is:

sound *frequency duration*

frequency is the pitch at which the sound is played. It's measured in Hertz (Hz), and the values for some musical notes are shown in Figure 6-1. To go up an octave, double the value; to go down an octave, halve it.

duration is the length for which the Sound command will play the *frequency*. *duration* is measured in milliseconds, or thousandths of a second. A value of 100 is equal to ⅒ second; a duration of 1000 is one second.

Use the Sound command to add individual and less annoying beeps to a batch file. For a quick beep to replace the seemingly eternal DOS beep, try:

```
sound 1760 100
```

Note	***Frequency***
C_5	523.25
$C\# - Db_5$	554.37
D_5	587.330
$D\# - Eb_5$	622.25
E_5	659.260
F_5	698.460
$F\# - Gb_5$	739.99
G_5	783.990
$G\# - Ab_5$	830.61
A_5	880.000
$A\# - Bb_5$	932.33
B_5	987.770

Figure 6-1. *Common frequencies in Hertz and their musical-note counterparts.*

You can also use several Sound commands to create interesting tunes. For example, consider the following Sound commands in a batch file:

```
sound 1174 100
sound 1318 100
sound 1048 150
sound 524 100
sound 784 200
```

This combination plays a little ditty that's bound to pull anyone in from outer space.

Using Batch Files in Batch Files

A batch-file program returns control to DOS after it carries out its last command, as do all programs (at least those that work). This means that if you put one batch file in the middle of another, the rest of the first batch file won't run. After the second batch file runs, it returns to DOS.

You can, however, invoke a batch-file program at the end of another batch file because there are no more commands to be carried out in the original batch file. Because you can use this technique to link a series of batch files, it's called *chaining* batch files.

If you really need to run a second batch file from within the first—and have it return—there's a special batch-file command that will do the job. This capability of running one batch file from another is referred to as *calling* because you *call* the batch file in to do something. Then, when it finishes, you continue with the business at hand. The command you use is, logically enough, called the Call command.

Even though the Call command was introduced in version 3.3 of DOS, you can still use this technique to make your batch files more flexible if you're using DOS 3.2 or an earlier version. Skip to the heading "The Command Command."

The Call Command

The Call command is used to run one batch-file program from within another. After the second batch file has run, control returns to the original batch file—just as if you were running any other program or DOS command. In fact, you can even use the Call command with some of those rare PC applications that will stop a batch file (such as some versions of WordStar).

The Call command has two parameters:

call *batchfile parameters*

batchfile is the name of the file of batch commands to be carried out.

parameters represents any parameters that *batchfile* might require. You can either enter the parameters themselves, or use replaceable parameters so that parameters typed with the name of the first batch file are passed along to the second batch file.

Call example

Here's an example of two short batch files, one of which uses the Call command to call the other. Make sure you're in the \BATCH directory, and copy from the console, or use Editor or your word processor, to create the files named FIRST.BAT and SECOND.BAT shown in Figure 6-2.

FIRST.BAT:

```
1: @echo off
2: for %%p in (%1) do call second %%p
3: echo *** Completed the jobs. ***
```

SECOND.BAT:

```
1: @echo off
2: echo Some useful processing by SECOND.BAT to %1.
```

Figure 6-2. *Using the Call command to call another batch command.*

The For command in line 2 of FIRST.BAT carries out a Call command for each member of %1 (most likely a file name with wildcards). The Call command, in turn, carries out SECOND.BAT, specifying the current value of %%p as a parameter. Rather than just look at it to see how it works, enter the following:

```
C>first *.*
```

DOS responds by displaying *Some useful processing by SECOND.BAT to* followed by the name of each file in the current directory. What you see on your screen will vary, but it might look something like this:

```
C>first *.*
Some useful processing by SECOND.BAT to PROMPTRS.BAT.
Some useful processing by SECOND.BAT to SECOND.BAT.
```

```
Some useful processing by SECOND.BAT to FIRST.BAT.
Some useful processing by SECOND.BAT to PLAY.BAT.
Some useful processing by SECOND.BAT to SOUND.EXE.
*** Completed the jobs. ***
```

When it has carried out SECOND.BAT for each file in the current directory, FIRST.BAT displays *** *Completed the jobs.* ***, and control returns to DOS.

In this example, SECOND.BAT simply echoes a message, but it could easily carry out a whole series of commands: printing files, copying them to storage, backing them up, and so on. Without the Call command in line 2 of FIRST.BAT—if it read simply *for %%p in (%1) do second.bat*—DOS would carry out SECOND.BAT once and then return to the system prompt.

A batch file can call itself

The ability to call one batch file from another is most valuable, especially when you write your own menu programs. But this ability doesn't come without cost. In the previous example, you used two batch files to do one job. Each file requires the minimum disk-storage space, and you have to keep track of both of them. But it isn't necessary to call a different batch file; a little sleight of hand with a parameter allows you to combine them into a single file that calls itself to accomplish the same thing (and without a busy signal).

Look at the batch file SELF.BAT in Figure 6-3. As you can see, lines 1, 3, and 4 are almost identical to FIRST.BAT in the previous example, and line 7 is the heart of SECOND.BAT. But lines 2 and 5 are new, there are a couple of labels that weren't there before, and there's a parameter called *trick* at the end of line 3. Can you see what's happening before running the program?

```
1:    @echo off
2:    if "%2"=="trick" goto DO_IT
3:    for %%p in (%1) do call self %%p trick
4:    echo *** Completed the jobs. ***
5:    goto END
6: :DO_IT
7:    echo Some useful processing to %1.
8: :END
```

Figure 6-3. *SELF.BAT: A batch file that calls itself.*

Here's what SELF.BAT does when you type *self* *.*:

1. The If command in line 2 doesn't find a second parameter at all, let alone *trick*, so the Goto command isn't carried out.

2. The For command in line 3 carries out a Call command for each file in the current directory (%1 is *.*). The Call command, in turn, loads and runs the SELF batch command with two parameters: The first is the file name and the second is *trick*. (Are you beginning to get the picture?)
3. The If command in line 2 of the copy of SELF.BAT called by the Call command finds that %2 is, indeed, *trick*, so it carries out the *goto DO_IT* command.
4. Line 7 of SELF.BAT (called by the Call command) echoes the message, and then this copy of SELF.BAT ends, returning to the For command in the first copy of SELF.BAT. This process of calling another copy of SELF.BAT with two parameters is repeated for each file in the current directory.

Create and save SELF.BAT, then type *self* *.*. The result should be the same as when you first typed *.* in the previous example. You have accomplished the same task using one file instead of two. This technique is used in a batch file to print a series of files in Chapter 12.

The Command Command

Call was created to let one batch file call another, but earlier versions of DOS let you do the same thing using the Command command. Note that if you're using DOS 3.3 or a later version of DOS, you can use the Call command instead of the Command command, described here. Even so, you might find the following description of the Command command interesting. If not, skip to the heading "Time Stamping Files, Date Stamping Files, and Other Output."

COMMAND.COM is the DOS command processor, the program that checks what you type and passes your instructions along to DOS. It's automatically loaded and executed when you start DOS. You can run this program, like any other file whose extension is COM, by typing its name. And you refer to the program by its name, like other DOS commands, so it's called the Command command.

Typing *command* loads and executes another copy of COMMAND.COM. The first copy—the one that looked at the Command command you entered—isn't gone; it's put into a sort of suspended animation and springs back to life when you end the new command processor with an Exit command. This all might seem unnecessarily convoluted, so try a couple of quick examples.

Enter the following:

```
C>command
```

DOS responds by loading another copy of COMMAND.COM, which displays the startup message you normally see when you first start the system.

Enter a Directory command:

```
C>dir
```

DOS reacts to commands as it always does. Now enter an Exit command to end this copy of COMMAND.COM:

```
C>exit
```

DOS displays the system prompt without appearing to do anything else, but now you're back to the original copy of COMMAND.COM.

❑ **NOTE:** *Some application programs, such as Microsoft Word, let you enter a single DOS command from within the application. If you enter a Command command, DOS displays the startup message and it looks as though you have exited from the application and returned to DOS; you can enter commands, change directories, and in general do whatever you like. But all of this is happening with the second copy of COMMAND.COM. The first copy of COMMAND.COM is still in memory, waiting for the second copy to end. To return to your application, end the second copy of COMMAND.COM with an Exit command. Be careful not to change or delete any files the application program is using, or the program might become confused and you'll have to restart your system.*

You can also load another copy of COMMAND.COM to execute a command and return to the previous copy of COMMAND.COM without entering an Exit command. This form of the Command command has two parameters:

command /c *program*

/c tells DOS to carry out *program* and return to the previous copy of COMMAND.COM.

program is any command, including a batch command or even another Command command.

This is the form of the Command command that lets you run secondary batch files from within another batch file. You type the name of the batch file—and any parameters it requires—following the /c parameter.

For example, you would enter the following to load another copy of COMMAND.COM, display the directory in wide format, and return to the current command processor:

```
C>command /c dir /w
```

The directory is displayed by the second copy of COMMAND.COM; the system prompt that follows it is displayed by the first.

There's still no free lunch

As you'll see in the following examples, using COMMAND.COM this way lets you write more flexible and powerful batch files. This power doesn't come without some penalty: Loading another copy of COMMAND.COM takes around 3000 bytes of memory and adds to the time required to carry out a batch command.

But 3000 bytes is a small price to pay for the additional capability, and COMMAND.COM loads fairly quickly from a hard drive. It takes longer to load from a floppy disk, of course, but you can minimize the penalty by loading it from a RAM disk. (See Chapter 10 for more information about using a RAM disk.)

Command examples

The DOS environment, which is described in Chapter 8, controls such characteristics as the search path and system prompt. Each copy of COMMAND.COM starts out with an environment identical to the previous COMMAND.COM; each inherits those items in the environment from the copy that loaded it. However, if you change the environment of a copy of COMMAND.COM, the changes are lost when it ends and returns to the copy that loaded it. This is easy to demonstrate. Load a new command processor by entering:

```
C>command
```

DOS displays the startup message. Enter the following command to change the system prompt:

```
C>prompt DOS here:
DOS here:_
```

DOS displays the new system prompt. Now enter an Exit command to end this copy of COMMAND.COM:

```
DOS here:exit

C>_
```

DOS displays the original system prompt because you have returned to the previous copy of COMMAND.COM, whose environment wasn't modified. It's fairly simple to check the amount of memory each additional copy of COMMAND.COM requires with the Check Disk (chkdsk) command. (If you have DOS 4.0 or later, you can use the Mem command; but remember the Command command is a shortcut for earlier versions of DOS, which lack the Call and Mem commands.)

First, enter the following to see how much memory is available:

```
C>chkdsk
```

The last line of the response tells you how much memory is available. Now enter a Command command to execute the Check Disk command:

```
C>command /c chkdsk
```

The number in the last line of this response should be about 3000 less than the number in the previous example; the actual difference is how much memory COMMAND.COM requires. You can load as many copies of COMMAND.COM as you like; enter the following to load two copies, the second of which executes the Check Disk command:

```
C>command /c command /c chkdsk
```

This time the difference in memory available should be twice the previous difference because you loaded COMMAND.COM and then loaded it again. You're back to the original version because both Command commands included the /c parameter, so both returned to the previous copy.

Using the Command command in a batch file

The following is an example of using the Command command to call one batch file from within another—just as was done earlier with the Call command. Make sure you're in the \BATCH directory, and copy from the console, or use Editor or your word processor, to create the files named ONE.BAT and TWO.BAT shown in Figure 6-4.

ONE.BAT:

```
1: echo off
2: for %%p in (%1) do command /c two %%p
3: echo *** Completed the jobs. ***
```

Figure 6-4. *Using the Command command to call another batch command.* *(continued)*

Figure 6-4. *continued*

TWO.BAT:

```
1: echo off
2: echo Some useful processing to %1.
```

The For command in line 2 of ONE.BAT carries out a Command command for each member of %1 (most likely a file name with wildcards). The Command command, in turn, carries out TWO.BAT, specifying the current value of *%%p* as a parameter.

To test these files, enter:

```
C>one *.*
```

DOS responds by displaying two lines for each file in the \BATCH directory: the system prompt followed by *Some useful processing to* followed by the file name. DOS loads and executes TWO.BAT once for each file in the root directory; if you're using floppy disks, you can hear DOS load the file each time. When it has carried out TWO.BAT for each file in the root directory, ONE.BAT displays **** Completed the jobs. **** and returns to DOS.

In this example, TWO.BAT simply echoes a message, but it could carry out a whole series of commands. Without the Command command in line 2 of ONE.BAT—if it read simply *for %%p in (%1) do two.bat*—DOS would carry out TWO.BAT once and return to the system prompt.

A batch file can call itself

In the previous example, you used two batch files to do one job. Each file requires the minimum disk-storage space, and you have to keep track of both of them. But it isn't necessary to call a different batch file; like the Call command, the Command command has a parameter that allows you to combine them into a single file that calls itself to accomplish the same thing.

Look at the batch file in Figure 6-5. As you can see, lines 1, 3, and 4 are almost identical to ONE.BAT in the previous example, and line 7 is the heart of TWO.BAT. But lines 2 and 5 are new, there are a couple of labels that weren't there before, and there's a parameter called *trick* at the end of line 3.

Here's what SELF.BAT does when you type *self *.**:

1. The If command in line 2 doesn't find a second parameter at all, let alone *trick*, so the Goto command isn't carried out.

2. The For command in line 3 carries out a Command command for each file in the \BATCH directory (%1 is *.*). Each Command command, in turn, loads and carries out the Self batch command with two parameters: The first is the file name and the second is *trick*.

3. The new copy of COMMAND.COM loads and runs SELF.BAT. This time the If command in line 2 finds that %2 is, indeed, *trick*, so it carries out the goto DO_IT command.

4. Line 7 echoes the message, and this copy of SELF.BAT ends, returning to COMMAND.COM. This copy of COMMAND.COM ends (the /c parameter says carry out just one command), returning to the previous copy of COMMAND.COM, which is still in line 3 of SELF.BAT with a list of file names. It repeats steps 3 and 4 for each file.

Create and save SELF.BAT, then type *self* *.*. The result should be the same as when you typed *one* *.* in the previous example. You have accomplished the same task using one file instead of two.

```
1:    echo off
2:    if "%2"=="trick" goto DO_IT
3:    for %%p in (%1) do command /c self %%p trick
4:    echo *** Completed the jobs. ***
5:    goto END
6: :DO_IT
7:    echo Some useful processing to %1.
8: :END
```

Figure 6-5. *SELF.BAT: A batch file that calls itself.*

Redirecting the Output of a Batch Command

If you have ever tried to redirect the output of a batch command, you've discovered that you can't. In the previous example, suppose you typed the following:

```
self *.* > self.out.
```

What DOS would do is to create a file named SELF.OUT—but it wouldn't write anything in it. The file's length would be 0.

You can, however, redirect the output of the Command command. If you entered SELF.BAT, enter:

```
C>command /c self *.* > self.out
```

DOS churns away but doesn't display anything, and finally displays the system prompt. Now enter:

```
C>type self.out
```

There's the output of the batch command, redirected to SELF.OUT. Erase it by entering:

```
C>erase self.out
```

Time Stamping Files, Date Stamping Files, and Other Output

You can use DOS to put the current time and date in a file or on printed output. Because this is like using a rubber stamp to put the time and date on a piece of paper, it's often called *time stamping* or *date stamping*.

The technique involves redirecting the input of a Time or Date command from a file that contains only a carriage return, and redirecting the output to the file or device that is to be stamped. DOS responds to the carriage return by leaving the time or date unchanged.

First, create a file that contains a carriage return by entering:

```
C>copy con cr.dat
<Enter>
^Z
```

Now type the following Date command to see how the technique works:

```
C>date < cr.dat
Current date is Wed 10-16-1991
Enter new date (mm-dd-yy):

C>_
```

But you don't want the second line in a file, of course, so pipe the output of the Date command through the Find filter, and redirect the final output to a file named DATE.TMP. It sounds complex, but enter the following:

```
C>date < cr.dat ¦ find "C" > date.tmp
```

Now verify the date stamp by displaying DATE.TMP:

```
C>type date.tmp
Current date is Wed 10-16-1991

C>_
```

To put a time or date stamp at the end of a file, use two > symbols to redirect the output of the Time or Date command and add it to the target file. Create a test file, and then put a time stamp at the end by entering:

```
C>copy con file1.doc
This is a test file.
^Z

C>time < cr.dat ¦ find "C" >> file1.doc
```

Putting a time or date stamp at the beginning of a file requires two temporary files: One is the time or date stamp itself, and the other is the combination of the time or date stamp and the original file. You already created DATE.TMP, which contains the date stamp. FILE1.DOC is a test file that has a time stamp at the end. Enter the following to combine DATE.TMP and FILE1.DOC, putting a time stamp at the beginning, and to display the resulting temporary file named DATE.$$$:

```
C>copy date.tmp+file1.doc date.$$$ > nul

C>type date.$$$
Current date is Mon 10-16-1989
This is a test file.
Current time is 4:44:12.68p
```

Redirecting the output of the Copy command to NUL means you don't see the messages that acknowledge the names and number of files copied. If you used this technique in a batch file to add a date or time stamp at the beginning of FILE1.DOC, you would erase the original (FILE1.DOC) and rename DATE.$$$ to FILE1.DOC.

To clean up your directory, enter the following:

```
C>erase date.$$$

C>erase date.tmp

C>erase file1.doc
```

❑ **NOTE:** *Don't erase CR.DAT! Keep it around so that you can use Time or Date this way again. Whenever you redirect the input of either the Time or Date command, make certain that the file from which you redirect the input contains only a carriage return. If it doesn't contain a carriage return, or if it contains an invalid reply to the Time or Date command, DOS will wait for the proper input from the file. And wait. And wait. And eventually you'll have to restart DOS.*

How long did that take?

Figure 6-6 shows a batch file, TIMER.BAT, that uses time stamps to record how long DOS takes to carry out a command. The technique is simple: It displays the current time, carries out the command, and redisplays the time.

```
1: @echo off
2: echo *** Starting time
3: time < cr.dat : find "C"
4: %1 %2 %3 %4 %5 %6
5: echo *** Ending time
6: time < cr.dat : find "C"
```

Figure 6-6. *TIMER.BAT: A DOS stopwatch.*

The only line of note is line 4: It contains nothing but replaceable parameters. This line is the event being timed. Including this many replaceable parameters lets you time almost any command.

Enter TIMER.BAT, and then test it by entering:

```
C>timer vol
*** Starting time
Current time is 5:09:34.69p
Volume in drive C is FORSYTHWOLF
*** Ending time
Current time is 5:09:35.82p

C>_
```

The difference between the starting and ending times in this example is 1.13 seconds. Your system might be faster or slower, depending on what type of computer you have.

TIMER.BAT can show you some interesting things about your system. Typing *timer*, for example, shows you how long TIMER.BAT itself takes to do

its work because there's no command for DOS to carry out. Try it. If you have a RAM disk, make it the current drive, copy TIMER.BAT and CR.DAT to it, and type *timer* again to see what difference using a RAM disk makes. Now copy FIND.EXE to the RAM disk and type *timer* again; you should see a substantial difference. This batch file can be a handy tool for measuring the speed of your system.

Using ANSI for Good Presentation

The power of ANSI.SYS is often overlooked by batch-file programmers. One problem is that you're not always sure a system has the ANSI.SYS driver installed. Assuming it is, you can use the ANSI codes to control the display, giving your batch files a near-professional look. One method of doing this ties directly into the Call command, discussed earlier in this chapter.

Throwing the ANSI attribute switches

Imagine a set of common batch files that control the display. Each would contain the ANSI codes required to create one particular display effect. Each effect could be identified by the batch file's name. Consider the following four batch-file examples (Figure 6-7):

NORMAL.BAT

```
 1: @echo [0m
```

BOLD.BAT

```
 1: @echo [1m
```

BLINK.BAT

```
 1: @echo [5m
```

REVERSE.BAT

```
 1: @echo [7m
```

Figure 6-7. *ANSI batch files that can be called by other batch files.*

Note how each batch file contains only the ANSI code to change one aspect of the display: NORMAL.BAT changes the display to normal; BOLD.BAT to bold or high intensity; BLINK.BAT to flashing characters;

and REVERSE.BAT to black characters on a white background. You could create additional batch files to change the color (BLUE.BAT, GREEN.BAT, and so on). Refer to Appendix A for the codes and their meanings.

You can use each of these in a batch file or on its own to control the text display. But if you use one in a batch file, remember to use the Call command first (or the Command command if you have an earlier version of DOS). For example:

```
call blink
echo I'm flashing!
call normal
```

In the above three lines, the BLINK.BAT file is called first. That sets the text attributes to flashing. The next Echo command displays text that blinks. Finally, a call is made to the NORMAL.BAT file, which resets the display back to standard white text on black background. (Always remember to do that, or you leave your user with a flashing or otherwise distorted DOS prompt!)

Moving text around

Using the Call command with various ANSI display-effect batch files seems like enough. But why not take full advantage of the power of ANSI.SYS? Consider a batch file that would display text at any given location on the screen, using the ANSI Locate Cursor command. Such a batch file is LOCATE.BAT, shown in Figure 6-8:

```
1: @rem %1=row, %2=col, %3...=string
2: @echo [%1;%2H %3 %4 %5 %6 %7 %8 %9
```

Figure 6-8. *The LOCATE.BAT file, which can be called by other batch files.*

In line 2 of Figure 6-8, you see how the ANSI Locate Cursor command is used; it gets its row and column parameters from the calling batch file as %1 and %2, respectively. The rest of the parameters, %3 through %9, represent the string to display. This limits you to a string of only seven words, but that's enough for most fancy displays. The end result is a batch file that displays a given string at a given location and that can be called by other batch files.

Bringing it all together

To put all these elements to work, recall the helpful text file discussed in Chapter 3. The file MONTHRPT.HLP contained ANSI codes that positioned the cursor, changed the display attributes, and created a more

interesting display than a plain text file would have. For practical purposes, that method is really the best way to display lots of text: Just stuff it all into a text file and use the Type command in a batch file. But since ANSI commands and calling batch files are the subjects here, consider the following batch file (Figure 6-9) that produces more or less the same display:

```
1: @echo off
2: cls
3: call bold
4: call locate 5 10 PREPARING THE MONTHLY REPORT
5: call normal
6: call locate 10 20 1. Enter data into spreadsheet
7: call locate 12 20 2. Update WP document named REPORT.DOC
8: call locate 14 20 3. Type
9: call bold
10: call locate 14 28 doreport
11: call normal
12: call reverse
13: call locate 20 10 Send printer output to department office
14: call normal
```

Figure 6-9. *The HELP.BAT file, which puts the Call command and ANSI batch files to work.*

The batch file HELP.BAT is nearly all Call commands. Each line from 3 onward uses the Call command to execute ANSI commands held in other batch files. Note how the LOCATE.BAT command is used.

If you enter and run this batch file, make sure all the other batch files are in the current directory or on your system's search path. The visual display will be fairly close to that achieved in the text file MONTHRPT.HLP. It's not as elegant, but it's essentially the same end result.

For additional sprucing up, you can add extended graphics characters to draw frames around related information or to represent keys, or create batch files that use ANSI commands to draw boxes at specific locations on the screen. It's possible to go overboard with this, but at least the DOS tool chest isn't as empty as some people believe.

All these features make your batch files longer, but the added readability and convenience almost always make it worth the effort—especially if you use many batch files or if your batch files are used by others. The batch files in the remainder of the book use these techniques and more to make them both powerful and easy to use.

Chapter Summary

- Design your batch files, don't just write them.
- Display instructions if the batch file needs parameters or if using it incorrectly could erase valuable files.
- Make your batch files easy to use by keeping the screen uncluttered and displaying helpful or reassuring messages.
- Don't use redirection symbols (<, >, or |) in a batch file for any purpose other than redirection.
- The Call command and Command command let you run secondary batch files just as you use DOS commands in a batch file.

CHAPTER

7

Designing an Interactive Menu System

A menu system tailored to the programs you use makes your computer a much more productive tool. A properly designed menu system can eliminate most of the time you spend changing directories and starting programs. You don't have to buy a menu program; DOS itself includes everything you need to build a rudimentary menu system. And when you add a short program called REPLY.COM—which you'll create in this chapter—you can make your menu system as interactive as you like.

In addition to showing you how to create REPLY.COM, this chapter shows the basic techniques of creating menus, checking responses, and displaying help information. Chapter 15 uses the techniques described here, plus batch programs described in following chapters, to create a multilevel menu system that will handle several application programs.

Making Batch Commands Interactive: REPLY.COM

Batch-file programs can be extremely useful and can streamline many operations, but one thing they aren't is interactive. The traditional way to communicate with a batch file is via the replaceable parameters typed after the batch-file name on the command line. This rules out on-the-fly decision making when a batch file is running. Yet there are ways to pass information to a batch file after it's started to run.

When some programs quit, they return a value to DOS. That value is known in batch file-speak as an *errorlevel*. For example, most programs run successfully and return an errorlevel of 0 to DOS. If they don't, they can return an errorlevel value, ranging from 1 through 255, to represent various things that went wrong. Your batch file can use the If command to interpret the errorlevel returned by the program, and then various actions can take place based on the result of the comparison.

REPLY.COM is a short program—a utility—that waits for a key to be pressed. It then identifies that key by returning its ASCII value as an errorlevel. From there, your batch file can examine the errorlevel value (which is really an ASCII keystroke) and interpret which key was pressed using the If command.

Because the If command checks for an errorlevel equal to or greater than the number you specify, you must check for key codes in descending order, starting with the highest and proceeding to the lowest. If the key codes you're looking for aren't in a single sequence, you also must be sure to check for the gaps. The reason for this, and the method, will become clear after a few examples.

You can look up the ASCII code returned by keys pressed in the standard typewriter portion of the keyboard in Appendix C. The extended-key code of the special keys—the function keys, keypad keys, and combinations that use the Alt and Ctrl keys—is two numbers, as shown in Appendix D. Because the first number of an extended key code is always 0, REPLY.COM sets *errorlevel* to the second number.

You should note that some ASCII codes are the same as the second number of the extended-key code of other keys. This creates some duplications: The semicolon key and the F1 function key, for example, both set *errorlevel* to 59; Alt-B and 0 (zero) both set *errorlevel* to 48. This shouldn't create a problem, because the duplicated keys are generally unrelated and won't interfere with each other.

Creating REPLY.COM with Debug

Chapter 5 describes how to use Debug to create a program. If you haven't gone through the examples there, you should probably do so before continuing with this section.

To start the Debug program and create REPLY.COM, enter:

```
C>debug
-a 100
```

A is the Assemble command, which invokes Debug's miniassembler program. Here you specified a value of 100 (hexadecimal) after A, indicating that you want to begin writing the program at memory address 100. Debug responds with an address *xxxx:0100*.

Now enter the following statements after each address:

```
xxxx:0100 mov ah,8
xxxx:0102 int 21
xxxx:0104 cmp al,0
xxxx:0106 jnz 10a
xxxx:0108 int 21
xxxx:010A mov ah,4c
xxxx:010C int 21
xxxx:010E <Enter>
```

Enter the following Unassemble command to display the program, and proofread the result against Figure 7-1:

```
-u 100 10d
```

Debug responds with:

```
xxxx:0100 B408          MOV     AH,08
xxxx:0102 CD21          INT     21
xxxx:0104 3C00          CMP     AL,00
xxxx:0106 7502          JNZ     010A
xxxx:0108 CD21          INT     21
xxxx:010A B44C          MOV     AH,4C
xxxx:010C CD21          INT     21
```

Figure 7-1. *Unassemble command output for REPLY.COM.*

If your output doesn't agree with Figure 7-1, go back to the heading "Creating REPLY.COM with Debug," enter the program, unassemble it, and proofread it again.

If your output agrees with Figure 7-1, enter the following commands to store REPLY.COM on disk and quit Debug:

```
-r cx
CX 0000
:e
-n reply.com
-w
Writing 000E bytes
-q
```

You should be back in DOS with the system prompt on the screen.

A quick test of REPLY.COM

Remember, REPLY.COM sets the errorlevel equal to the key code of the key that is pressed. If you don't include an If command that checks for errorlevel, REPLY.COM behaves just like the Pause command—halting processing until a key is pressed—without the *Press any key to continue* message. The first test of REPLY.COM is easy—just type:

```
C>reply
```

and press Enter.

Nothing should happen. In particular, there should be no system prompt. (If there is a system prompt, type *reply* again. If the system prompt appears again, go back to the heading "Creating REPLY.COM with Debug" and enter the program again. If there's a more serious problem with the program, you may have to restart the system by pressing Ctrl-Alt-Del or even by turning the power off and on.)

You won't see a system prompt because REPLY.COM is waiting for you to press a key. Now press any key. DOS should display the system prompt again. This may seem like a trivial job for a program to do, but even this limited action is useful, because it lets you pause in a batch file without displaying the *Press any key to continue* message. However, this is hardly the reason REPLY.COM is so valuable.

There's no sign of whether REPLY.COM set *errorlevel* to the code of the key you just pressed, but it's easy to test with a short batch file.

Using REPLY.COM to accept any key

The first demonstration of REPLY.COM is a batch file called MENU-1.BAT that displays a two-item menu and waits for you to reply. If you press F1, MENU-1.BAT clears the screen. If you press F2, it displays the directory. If you press any other key, the program returns to DOS.

Using Editor, Edlin, or a word processor, enter MENU-1.BAT as shown in Figure 7-2. (You could copy from the console, but the Copy command doesn't let you go back and change a line after you have pressed Enter, and this file is a bit long to type perfectly.)

The labels in MENU-1.BAT don't have to be in uppercase, and you don't have to enter the blanks at the beginning of the lines that aren't labels. These conventions make it easier to see how the batch file works, but they don't affect what the batch file does. The line numbers are for reference only.

```
 1:     @echo off
 2:     cls
 3:     echo F1 - Clear the screen
 4:     echo F2 - Display the directory
 5:     echo Press any other key to quit
 6:     reply
 7:     if errorlevel 61 goto END
 8:     if errorlevel 60 goto F2
 9:     if errorlevel 59 goto F1
10:     goto END
11:  :F1
12:     cls
13:     goto END
14:  :F2
15:     dir
16:  :END
```

Figure 7-2. *MENU-1.BAT: A short menu.*

Here's how MENU-1.BAT works:

- Lines 1–6 clear the screen, display the menu, and run REPLY.COM.
- Line 7 checks whether the key code is equal to or greater than 61 (to see whether a function key other than F1 or F2 was pressed). Because the If command checks whether *errorlevel* is equal to or greater than the specified number, you must check from the largest value to the smallest. MENU-1.BAT does something only if F1 or F2 is pressed, so this If command says don't do anything (*goto END*) if the key that was pressed was F3 or greater.
- Line 8 says if the key code is 60 (function key F2 was pressed), go to the label F2. Line 7 eliminated the possibility that any key whose code is higher than 60 was pressed, so this If command isolates the F2 key.
- Line 9 says if the key code is 59 (function key F1 was pressed), go to the label F1. Lines 7 and 8 eliminated the possibility that any key whose code is higher than 59 was pressed, so this If command isolates the F1 key.
- Line 10 says go to the label END, exiting the batch file without doing anything further. Just as the If command in line 7 ignores all keys higher than F2, this If command ignores all keys lower than F1.
- Lines 11–13 define what happens if the F1 key is pressed: The screen is cleared and control returns to DOS with *goto END*.
- Lines 14–16 define what happens if the F2 key is pressed: The directory is displayed and control returns to DOS.

Save MENU-1.BAT. Now run *menu-1*. The screen will clear and you'll see your menu:

```
F1 - Clear the screen
F2 - Display the directory
Press any other key to quit
_
```

Press F1. The batch file should clear the screen and return control to DOS. Enter *menu-1* again to run MENU-1.BAT, then press the F2 key. The batch file should display the directory. Try it a few more times, pressing other keys to make sure that any key other than F1 or F2 simply returns to DOS. (This is a part of *bulletproofing*—making sure the program really works.)

Using REPLY.COM to accept only certain keys

MENU-1.BAT accepts any key but acts on only F1 and F2; it returns to DOS if any other key is pressed. This isn't always the best way to handle replies,

especially if an incorrect response can damage something, such as erase a file. You can modify MENU-1.BAT so it accepts only F1 or F2 and waits to reply until one of those keys is pressed.

Figure 7-3 shows the revised version of MENU-1.BAT that ignores all keys except F1 and F2. The changes are shown in italics. Make a copy of MENU-1.BAT named MENU-2.BAT. Now use Editor or your word processor to make the changes and save the revised file as MENU-2.BAT.

The changes are minor:

- In line 5, change *Press any other key to quit* to *Press F1 or F2*.
- Insert a new line 6, which is the label *:REPLY*.
- In lines 8 and 11 (lines 7 and 10 in Figure 7-2), change END to REPLY. Now, when you press any key other than F1 or F2, the batch file goes back to run REPLY.COM again instead of returning to DOS. The only way to end the batch file is to press F1 or F2 (or Ctrl-C to cancel it).

Make the changes and save the revised batch file as MENU-2.BAT. Now run *menu-2*. You should see your menu on the screen, but with a different instruction line:

```
F1 - Clear the screen
F2 - Display the directory
Press F1 or F2
_
```

```
1:     @echo off
2:     cls
3:     echo F1 - Clear the screen
4:     echo F2 - Display the directory
5:     echo Press F1 or F2
6:   :REPLY
7:     reply
8:     if errorlevel 61 goto REPLY
9:     if errorlevel 60 goto F2
10:    if errorlevel 59 goto F1
11:    goto REPLY
12:  :F1
13:    cls
14:    goto END
15:  :F2
16:    dir
17:  :END
```

Figure 7-3. *MENU-2.BAT: A short menu that requires a correct response.*

The F1 and F2 keys should behave as they do in MENU-1.BAT, but if you press any other key, nothing should appear; the batch file simply waits for you to press either F1 or F2.

REPLY.COM can be used in any type of batch file, not just one that builds menus. For example, if you're preparing to format a floppy disk, you might create a batch file that issues a prompt that says Press F1 to format the disk; any other key cancels the formatting request, reducing the possibility of inadvertently formatting the wrong disk. You'll find batch files in this book that use both approaches to a keypress not specified in the batch file.

Using REPLY.COM to isolate a single key

You can isolate a single key entered in response to REPLY.COM with an If command that includes another If command. Type the following to create a file named MENU-3.BAT:

```
C>copy con menu-3.bat
@echo off
echo Press F1 to display the directory; any other key quits.
reply
if errorlevel 59 if not errorlevel 60 dir
^Z<Enter>
```

The first If command selects all keys whose code is equal to or greater than 59, and the second If command selects all keys whose code is less than 60. Only key code 59—function key F1—passes both tests. Run *menu-3*; if you press F1, DOS displays the directory. If you press any other key, DOS simply displays the system prompt. To generalize: The following If command carries out *command* if the key whose code is *x* is pressed in response to REPLY.COM:

if errorlevel *x* if not errorlevel *x*+1 *command*

if errorlevel x is true only for *x* and all values greater than *x*. The second part, *if not errorlevel x+1*, is true only if the value is not greater than *x*. And the final part, *command*, is executed only if errorlevel is equal to *x*. This technique is used to process responses to REPLY.COM in batch files in the remainder of the book.

Displaying Additional Information with a Batch File

Sometimes a menu or other screen might justify additional instructions—especially if you don't see it often or if someone less experienced with DOS will see it—but you'd rather keep the screen as uncluttered as possible.

Why not use the same technique that many application programs use: Display help information only if a particular key is pressed. You can do this by modifying MENU-2.BAT so that pressing Alt-H displays additional instructions.

> ❑ **NOTE:** ␛ *in Figure 7-4 and the following figures represents the Escape character (ASCII 27). If you're using Editor (or any WordStar-like editors), press Ctrl-P and then press the Esc (Escape) key; in Edlin, enter the Escape character by pressing Ctrl-V and then a left square bracket ([); if you're using Microsoft Word, press Alt-27; in WordPerfect, press Ctrl-V and then Escape.*

Figure 7-4 shows a revised version of MENU-2.BAT. The revised batch file carries out several Echo commands that display additional information if Alt-H is pressed; the changes to MENU-2.BAT are in italics.

Make a copy of MENU-2.BAT and use Editor, Edlin, or your word processor to make the changes, then save the revised file as MENU-4.BAT.

```
 1:    @echo off
 2:    cls
 3:    echo F1 - Clear the screen
 4:    echo F2 - Display the directory
 5:    echo Press F1 or F2 (or Alt-H for help)
 6: :REPLY
 7:    reply
 8:    if errorlevel 61 goto REPLY
 9:    if errorlevel 60 goto F2
10:    if errorlevel 59 goto F1
11:    if errorlevel 35 if not errorlevel 36 goto HELP
12:    goto REPLY
13: :F1
14:    cls
15:    goto END
16: :F2
17:    dir
18:    goto END
19: :HELP
20:    echo ␛[18;50HThis information is displayed
21:    echo ␛[19;50Honly when Alt-H is pressed.
22:    echo ␛[20;50HThere is no other effect on
23:    echo ␛[21;50Hthe batch file.
24:    goto REPLY
25: :END
```

Figure 7-4. *MENU-4.BAT: A batch file that displays help information.*

This example uses ANSI escape sequences, so make sure you have the ANSI.SYS device driver installed. (For more information about ANSI.SYS, see Chapter 3.)

Again, there aren't many changes to the file:

- In line 5, add the phrase (*or Alt-H for help*).
- Insert line 11, which goes to the label *HELP* if Alt-H (key code 35) is pressed.
- Insert lines 18–24. The message displayed by each Echo command begins with an ANSI.SYS command that positions the cursor, starting in row 18, column 50. (ANSI.SYS commands are described in Chapter 3.)

Make the changes and save the revised batch file as MENU-4.BAT. Now run *menu-4*. The instruction line includes an additional option:

```
F1 - Clear the screen
F2 - Display the directory
Press F1 or F2 (or Alt-H for help)
_
```

F1 and F2 work just as they did in MENU-2.BAT, but press Alt-H. MENU-4.BAT displays the help information at the lower right, then goes back to wait for you to press F1 or F2.

You can elaborate the help information by using display attributes, such as reverse video or color, or by enclosing the information in a box using the IBM extended characters. You don't need to get fancy, however, to display additional information on demand.

Erasing Part of the Display with a Batch File

Once you have displayed additional information with Alt-H, you might want to erase it. You can modify MENU-4.BAT so that the batch file displays the help information, then erases it when any key is pressed.

Figure 7-5 shows the revised version of MENU-4.BAT. It displays help information, then uses REPLY.COM to wait for you to press a key and, regardless of what key is pressed, erases the help information and goes back to wait for you to press F1 or F2. The changes are in italics.

The changes to the batch file are as follows:

- Insert lines 24–30:

- Line 24 adds the instruction *Press any key to continue*. The ANSI.SYS command ␛[7m starts reverse video; ␛[m turns off all attributes (returns to normal display).
- Line 25 waits for you to press a key.
- Lines 26–30 move the cursor to the beginning of each line of the help information and erase to the end of the line.

Make a copy of MENU-4.BAT named MENU-5.BAT. Use Editor, Edlin, or your word processor to make the changes to MENU-5.BAT, and save the revised file.

```
 1:    @echo off
 2:    cls
 3:    echo F1 - Clear the screen
 4:    echo F2 - Display the directory
 5:    echo Press F1 or F2 (or Alt-H for help)
 6: :REPLY
 7:    reply
 8:    if errorlevel 61 goto REPLY
 9:    if errorlevel 60 goto F2
10:    if errorlevel 59 goto F1
11:    if errorlevel 35 if not errorlevel 36 goto HELP
12:    goto REPLY
13: :F1
14:    cls
15:    goto END
16: :F2
17:    dir
18:    goto END
19: :HELP
20:    echo ␛[18;50HThis information is displayed
21:    echo ␛[19;50Honly when Alt-H is pressed.
22:    echo ␛[20;50HThere is no other effect on
23:    echo ␛[21;50Hthe batch file.
24:    echo ␛[22;50H␛[7mPress any key to continue.␛[m
25:    reply
26:    echo ␛[18;50H␛[K
27:    echo ␛[19;50H␛[K
28:    echo ␛[20;50H␛[K
29:    echo ␛[21;50H␛[K
30:    echo ␛[22;50H␛[K
31:    goto REPLY
32: :END
```

Figure 7-5. *MENU-5.BAT: A batch file that erases part of the screen.*

Now run *menu-5* and press Alt-H. The help information includes an additional line:

```
This information is displayed
only when Alt-H is pressed.
There is no other effect on
the batch file.
Press any key to continue.
```

Press any key. The help information is erased and the batch file waits for you to press F1 or F2 (or press Alt-H to display the help information again).

Adding more features like this to your batch file makes them longer, but the added convenience is worth it, especially if you have many batch files or if others use the batch files you write.

Changing Both the Display and Behavior of a Batch File

Because ANSI.SYS commands let you display information anywhere on the screen, you can not only display additional information in a batch file, you can change what is already displayed. For example, you could include more than one set of menu choices in a single batch file and switch back and forth between the alternate sets by pressing a specified key.

Figure 7-6 shows a batch file based on MENU-5.BAT that lets you change the menu choices by pressing the right-arrow key. It's somewhat longer than MENU-5.BAT, because there are two complete sets of commands that display choices, check your response, and carry out the appropriate commands.

Because the differences from MENU-5.BAT are substantial, it's probably easier to enter it from scratch. Create a file named MENU-6.BAT and enter the lines shown in Figure 7-6.

```
1:     @echo off
2:     cls
3:     echo F1 -
4:     echo F2 -
5:     echo Press F1 or F2 (or right arrow to change the choices)
6:  :OPTION_1
7:     echo [1;6H[KClear the screen
8:     echo [2;6H[KDisplay the directory
```

Figure 7-6. *MENU-6.BAT: A batch file that presents two sets of choices.* *(continued)*

Figure 7-6. *continued*

```
 9:  :REPLY
10:     reply
11:     if errorlevel 77 if not errorlevel 78 goto OPTION_2
12:     if errorlevel 61 goto REPLY
13:     if errorlevel 60 goto F2
14:     if errorlevel 59 goto F1
15:     goto REPLY
16:  :F1
17:     cls
18:     goto END
19:  :F2
20:     dir
21:     goto END
22:  :OPTION_2
23:     echo [1;6H[KDisplay the disk volume label
24:     echo [2;6H[KDisplay the DOS version
25:  :REPLY_2
26:     reply
27:     if errorlevel 77 if not errorlevel 78 goto OPTION_1
28:     if errorlevel 61 goto REPLY_2
29:     if errorlevel 60 goto F2_2
30:     if errorlevel 59 goto F1_2
31:     goto REPLY_2
32:  :F1_2
33:     vol
34:     goto END
35:  :F2_2
36:     ver
37:  :END
```

Here's how MENU-6.BAT works:

- Lines 1–5 clear the screen and display the parts of the menu that don't change. Notice that only the key names (F1 and F2) are permanently displayed; because the choices themselves can be changed, they are shown separately. To keep this example to a reasonable size, the instructions in line 5 include a description of how to change the choices so that no additional help information is needed.
- Lines 6–8 display the first set of menu choices. ANSI.SYS commands are used to position the cursor ([1;6H) and erase to the end of the line ([K) before displaying the choices. Erasing to the end of the line allows for alternate menu choices that are of different lengths.
- Lines 9–15 run REPLY.COM to wait for a key to be pressed and evaluate the response. These lines differ from MENU-5.BAT in two ways.

  - There's no check for Alt-H because what was previously help information is now a permanent part of the display.
  - Line 11 checks to see whether the right arrow (key code 77) was pressed; if it was, the batch file goes to the label OPTION_2 to display the other set of options.
- Lines 16–21 carry out the first set of menu choices. This is just like MENU-5.BAT.
- Lines 22–36 are almost the same as lines 6–21 but are related to the second set of menu options:
  - Lines 22–24 display the second set of menu options, just as lines 6–8 display the first set of menu options. The new choices described in lines 23 and 24 are carried out by the Volume and Version commands in lines 33 and 36.
  - Lines 25–31 wait for a key to be pressed and evaluate the response, just as lines 9–15 wait for and evaluate the response to the first set of options.
  - Lines 32–36 carry out the second set of menu options, just as lines 16–21 carry out the first set of menu options.

Enter the file and save it as MENU-6.BAT.

Now run *menu-6*. The menu includes the instruction for changing the menu choices:

```
F1 - Clear the screen
F2 - Display the directory
Press F1 or F2 (or right arrow to change the choices)
```

Press the right-arrow key. Now the display should read:

```
F1 - Display the disk volume label
F2 - Display the DOS version
Press F1 or F2 (or right arrow to change the choices)
```

Press F1. MENU-6.BAT should display the volume label. Run *menu-6* again, and this time press F1 without pressing the right-arrow key. MENU-6.BAT should clear the screen. Both sets of options are available from the same batch file.

This is a lot of work to go through to simply offer four choices, especially because they would be handled better by a single four-choice menu. But what if a menu offers 20 options? A batch file described in Chapter 12 does just that: It displays a menu for sending commands to the printer. The

batch file uses the technique described in this section to divide the menu choices into two manageable sets.

The examples in this chapter used a simple two-choice menu to illustrate some techniques for designing menus. Chapter 15 combines these techniques with many of the batch files described in following chapters to set up a system of menus and submenus that you can use for day-to-day operation of your system and all your application programs.

Security in Batch Files

Security is an important part of using a computer. It's not James Bond–type security, although you should protect sensitive data. Instead, security is simply being safe; it's doing a daily backup or having some regular routine for making backup copies of your data. In a batch file, you can take measures to guard against potential mistakes. In particular, you can use REPLY.COM or a program like it as a type of security before attempting a dangerous operation.

Using REPLY.COM in Sensitive Situations

Where information security is important, you can use REPLY.COM as a form of protection in your menus. For example, instead of using the *errorlevel* result from REPLY.COM to help select a menu item or run a program, the result can be used as a password as in the following batch file, Figure 7-7, a variation of MENU-1.BAT:

```
1:    @echo off
2:    cls
3:    echo F1 - Enter Payroll program
4:    echo F2 - Sales Record
5:    echo Press any other key to quit.
6:    reply
7:    if errorlevel 61 goto END
8:    if errorlevel 60 goto F2
9:    if errorlevel 59 goto F1
10:   goto END
11: :F1
12:   cls
13:   echo Press security key to pass...
14:   reply
15:   if errorlevel 1 if not errorlevel 2 echo OK!
```

Figure 7-7. *MENU-7.BAT, which adds password security to the batch file using REPLY.COM.* *(continued)*

Figure 7-7. *continued*

```
16:     rem Security key is Ctrl-A
17:     goto END
18:  :F2
19:     cls
20:     rem Sales Record would go here...
21:  :END
```

Make a copy of MENU-1.BAT, renaming it MENU-7.BAT. Then use Editor, Edlin, or your favorite word processor to add the lines above that are in italics.

Here's what's new:

- Lines 3 and 4 now display some options that could contain sensitive information. On a system used by several individuals, or maybe even as extra security on a single-user system, these are options you want protected. (The examples may be a bit extreme; this type of security can be applied to a wide range of circumstances.)
- Insert lines 13–16. These lines are responsible for entering the security code, which is simply a one-keystroke code that allows a user to continue. Here the keystroke tested for, via reply in line 14, is Ctrl-A, which is code 1. The program only displays *OK!* in line 15; in real life, it would probably execute some payroll or accounting program.
- Line 20 is simply a holding place—a Remark command—that would be replaced by a similar test such as that seen in lines 13 through 16.

Save the program as MENU-7.BAT and then run *menu-7* at the DOS prompt. Press F1 to run the Payroll program. Then press the secret code, which is Ctrl-A.

This way of using REPLY.COM is interesting—and it works. But it's rather naive. An astute user, the kind that would want to peek into a payroll program, would simply use the Type command to list the batch file on the screen, then see what the secret control code is, or even just see which program is running, and avoid the batch file altogether. Basically, it boils down to this: There really isn't any security in a batch file, or in any personal computer running DOS. But you can take steps to protect your data and to prevent accidental loss of information.

Stepping on the Breaks

Isn't *Break* a funny name for a key on a computer keyboard? When you think about it, the name should really be *Brake*, as in stop. The Break key,

or Ctrl-C (or Ctrl-Break), is the only way to stop a running batch file. After you press Ctrl-C (or Ctrl-Break), the batch file stops executing the current line, and you see the following displayed:

```
Terminate batch job (Y/N)?
```

Here, DOS allows you to press Y to continue with the batch file (at the following line; the current line isn't executed) or N to stop the batch file and return to DOS.

Often you actually want a user to press Ctrl-C to halt a batch file. The most common technique for achieving this uses the following two lines:

```
echo Press Ctrl-C to cancel or
pause
```

That first echo statement is a fragment: The question looms, or *what?* But the answer is supplied by the output of the Pause command. The two lines make sense together on screen:

```
Press Ctrl-C to cancel or
Press any key to continue...
```

Here, you're giving the user an opportunity to *break out* of the batch file, or stop it from running. The Pause command waits for the user to *Press any key to continue.* While it's doing that, the user can also press Ctrl-C to stop the batch file, see the *Terminate batch job (Y/N)?* message, and then return to DOS.

Such a technique comes in handy in many places. The most obvious are formatting or deleting files on disk, operations in which you must make doubly sure that the user wants to proceed before the batch file continues. But there are better, more elegant ways to hop or skip around batch files.

The ASK.COM Program

A common variant of the REPLY.COM program is ASK.COM. Like REPLY.COM, ASK.COM waits for you to press a key and then returns an errorlevel value depending on which key you pressed. But unlike REPLY.COM, ASK.COM looks for only two keys: Y and N. It then returns an errorlevel of either 0 or 1, depending on whether Y or N was pressed, respectively. You can use ASK.COM to ask a yes-or-no question and then allow the batch file to proceed based on the results. ASK.COM waits for either Y or N (or Ctrl-C) and won't continue until one or the other is pressed.

Creating ASK.COM with Debug

Earlier in this chapter, you created REPLY.COM by using Debug and its miniassembler. Now you're going to create ASK.COM—which works a lot like REPLY.COM—in the same way.

To start the Debug program, enter:

```
C>debug
-a 100
```

Just as before, use the A command to assemble the program starting at address 100 (hexadecimal). Debug returns the address *xxxx:0100*, which is where you'll start entering the instructions for ASK.COM.

ASK.COM is a bit longer than REPLY.COM, so double-check your typing. Carefully enter the following into Debug:

```
xxxx:0100 mov ah,08
xxxx:0102 int 21
xxxx:0104 and al,5f
xxxx:0106 cmp al,59
xxxx:0108 jz 0112
xxxx:010A cmp al,4e
xxxx:010C jnz 0100
xxxx:010E mov al,01
xxxx:0110 jmp 0114
xxxx:0112 mov al,00
xxxx:0114 mov ah,4c
xxxx:0116 int 21
xxxx:0118 <Enter>
```

Enter the following Unassemble command, and then proofread the result against Figure 7-8:

```
-u 100 116
```

Debug responds:

```
20E0:0100 B408          MOV     AH,08
20E0:0102 CD21          INT     21
20E0:0104 245F          AND     AL,5F
20E0:0106 3C59          CMP     AL,59
20E0:0108 7408          JZ      0112
```

Figure 7-8. *Unassemble command output for ASK.COM.* *(continued)*

Figure 7-8. *continued*

```
20E0:010A 3C4E          CMP     AL,4E
20E0:010C 75F2          JNZ     0100
20E0:010E B001          MOV     AL,01
20E0:0110 EB02          JMP     0114
20E0:0112 B000          MOV     AL,00
20E0:0114 B44C          MOV     AH,4C
20E0:0116 CD21          INT     21
```

If your output doesn't match Figure 7-8, quit Debug (with the Q command), go back to the heading "Creating ASK.COM with Debug," and start over. Otherwise, enter the following commands to save ASK.COM on disk and return to DOS.

```
-r cx
CX 0000
:18
-n ask.com
-w
Writing 0018 bytes
-q
```

You should be back in DOS with the system prompt on the screen.

A quick test of ASK.COM

The program you just created, ASK.COM, sets the *errorlevel* equal to 1 or 0 if either N or Y was pressed, respectively. This allows you to ask yes-or-no questions in a batch file and then proceed accordingly. But unlike REPLY.COM, with ASK.COM you must press either Y or N for the batch file to continue. Try it. At your DOS prompt, enter the following:

```
C>ask
_
```

Nothing should happen; there should be no system prompt, just a blinking cursor. (If there is a system prompt, or you see anything else, then you probably entered the program incorrectly. Return to the heading "Creating ASK.COM with Debug" and enter the program again.)

Right now, ASK.COM is waiting for you to press a key. But not just any key. Try pressing F1, Alt-H, A, B, C, Enter, and so on. ASK.COM just keeps waiting until you press either Y or N. Go ahead and press the Y or N key now. There's the DOS prompt. Try the program again, but this time use a capital Y or N. ASK.COM interprets both uppercase and lowercase letters as the same. (REPLY.COM does not.)

While the program does work here at the DOS prompt, the only way to make it useful is to put it into a batch file. ASKTEST.BAT, shown in Figure 7-9, shows how ASK.COM can be put to use.

```
 1:    @echo off
 2:    echo Do you want to proceed with the format? (Y/N)
 3:    ask
 4:    if errorlevel 1 goto NOPE
 5:    rem Y was pressed
 6:    echo Okay, formatting disk...
 7:    goto END
 8:  :NOPE
 9:    echo Whew!
10:  :END
```

Figure 7-9. *ASKTEST.BAT, which tests the ASK.COM program.*

Here's how ASKTEST.BAT works:

- Line 2 poses the serious question, *Do you want to proceed with the format? (Y/N).* In this fictional example, the question deals with formatting a disk. Note how the prompt shows *(Y/N).* This is a common shorthand in computer program interfaces, telling users they have a choice of pressing *Y* for Yes or *N* for No.
- Line 3 gets a response to the question using the ASK.COM file. ASK.COM returns an *errorlevel* of 1 if N is pressed, 0 if Y is pressed.
- Line 4 tests for errorlevel 1; if it is found, execution branches to the NOPE label (line 8). Otherwise, lines 5–7 assume Y was pressed, and the format (or whatever) proceeds. (Here, in line 6, only a harmless message is echoed to the display.)
- Lines 8 and 9 are executed when N is pressed. In this case, only the message *Whew!* is displayed, and then the batch file ends.

Enter this batch file into Editor, Edlin, or your word processor. Then save it as ASKTEST.BAT. Now run *asktest.* You'll see the following:

```
Do you want to proceed with the format? (Y/N)
```

Press Y or N. If you press N, you'll see *Whew!* displayed; if you press Y, you'll see *Okay, formatting disk.*

Using ASK.COM in AUTOEXEC.BAT

ASK.COM can be put to use in a batch file to set certain options or run parts of the batch file. This comes in especially handy in configuration

batch files, such as AUTOEXEC.BAT. Again, the object is to give users a choice and to allow them to skip over or avoid some potentially dangerous option. Using either ASK.COM or REPLY.COM, you can effectively meet that goal.

Consider, for example, an AUTOEXEC.BAT file that sets up the computer with the memory-resident Print command. (The Print command is covered in *Running MS-DOS* and is also mentioned in Chapter 12 of this book.) However, there may be a time when you're using DOS and need more memory. In that instance, it would be wise not to load the Print command into memory, reserving that memory for the programs that need it. Look at the AUTOEXEC.BAT file in Figure 7-10. (You don't need to enter this file.)

```
 1:    @echo off
 2:    rem autoexec.bat, 10/19/92
 3:    path=c:\dos;c:\batch;c:\
 4:    prompt $p$g
 5:    echo Do you want to load the Print command?
 6:    ask
 7:    if errorlevel 1 goto NOPRINT
 8:    print /d:lpt1 /b:8196 /s:25 /m:5 /u:2
 9: :NOPRINT
10:    ver
11: :END
```

Figure 7-10. *An AUTOEXEC.BAT file that uses ASK.COM.*

This AUTOEXEC.BAT program uses ASK.COM to help set up the system. Here's what's going on:

- Lines 1 through 4 are standard system setup commands you'll find in most AUTOEXEC.BAT files. First, echo is turned off in line 1; a Remark command in line 2 identifies and dates the file; then lines 3 and 4 set the search path and system prompt. (Those commands are covered later, in Chapter 8.)
- In line 5, the Echo command displays the question *Do you want to load the Print command?* In line 6, the ASK.COM program is used to elicit an answer, either Y or N.
- Line 7 interprets the key pressed in line 6. If Y was pressed, an errorlevel of zero is returned and line 8 will be executed next. If N was pressed, the instructions *goto NOPRINT* are executed.
- In line 8, the Print command is loaded.

- Line 9 is the NOPRINT label, followed by line 10, which displays the DOS version. Note that in all instances, line 10 is executed. ASK.COM is set up so that only the line that loads the Print command is skipped. (More advanced versions can have a Y answer execute one section of commands and N execute another. But that requires more Goto commands to skip over various lines in the batch file.)

Password Protection in a Batch File

Any user who knows DOS will easily be able to circumvent the feeble password protection a batch file offers. Also, REPLY.COM only allows you to enter one character as a "password." Another approach is to require users to enter a password after the batch file name at the DOS prompt. Even this has its drawbacks.

The batch file in Figure 7-11 tests for a password, which must be entered after the name of the batch file and is interpreted as parameter *%1*:

```
 1:    @echo off
 2:    if "%1"=="" goto ERROR
 3:    if "%1"=="farkle" goto FARKLE
 4:    if "%1"=="blort" goto BLORT
 5: :ERROR
 6:    echo Type your password after passwd and try again
 7:    goto END
 8: :FARKLE
 9:    echo Welcome, agent Farkle!
10:    goto end
11: :BLORT
12:    echo Welcome, agent Blort!
13: :END
```

Figure 7-11. *PASSWD1.BAT, a password-protected batch-file program.*

Create PASSWD1.BAT using a text editor or word processor, using the commands shown in Figure 7-11. Here's how the program works:

- Line 2 tests to see if a password was entered. If so, the parameter %1 would be equal to *something*. If not, execution branches to the ERROR label at line 5, the message *Type your password after passwd and try again* is displayed, and control returns to DOS.
- Lines 3 and 4 test several password possibilities. Here only two are tested, *farkle* and *blort*, although in real life they would probably be passwords for individuals or various tasks that the batch file can carry

out. In either instance, if a matching password is found by the If command, control branches to the appropriate label, FARKLE or BLORT. (More passwords could be added here.)

- Lines 8–10 are executed when the matching password *farkle* is entered. In the example, only a message is displayed. However, if this were an actual data-sensitive situation, whatever programs the password *farkle* activates would appear in these lines.
- Lines 11 and 12 do the same thing for the *blort* password, although in this limited example all that happens is a message is displayed.

Run this batch file. Try several combinations: type *passwd1* alone, follow it with one of the two passwords, make up a password, and so on.

A fatal flaw

There are two major drawbacks to this batch file. First, it's easy for anyone to type it out and see what the passwords are. Second, the program doesn't work when you enter a password in all caps or capitalize any part. The persnickety If command is case sensitive.

A common solution to the second problem is to list every combination of password possible. But consider that a five-letter word can have well over 50 different combinations of uppercase and lowercase letters. A human being can think that Blort and BLORT both match *blort*, but to a computer they're totally different.

A possible solution is shown in Figure 7-12. But this solution still has a problem. See if you can figure out what it is.

You can create PASSWD2.BAT by copying the contents of PASSWD1.BAT into it and then editing in the new line 3 using your text editor or word processor.

```
 1:    @echo off
 2:    if "%1"=="" goto ERROR
 3:    goto %1
 4: :ERROR
 5:    echo Type your password after passwd and try again
 6:    goto END
 7: :FARKLE
 8:    echo Welcome, agent Farkle!
 9:    goto end
10: :BLORT
11:    echo Welcome, agent Blort!
12: :END
```

Figure 7-12. *PASSWD2.BAT, which uses the Goto command to match passwords.*

The only new addition in PASSWD1.BAT is the *goto %1* command. This tells DOS to go to a label, but the label was supplied when the program was started. Why does this work? Because unlike the If command, Goto doesn't care about the case of its labels.

Save the program on disk and then run it. Try all of the following:

```
passwd2 farkle
passwd2 FARKLE
passwd2 Farkle
```

And so on. They'll all work. But what is the fatal flaw? Try typing in a nonsense password. For example:

```
C>passwd2 piggy
Label not found
```

Aha! The problem is that the Goto command is always expecting the *%1* replaceable parameter to be a label. If you enter an unknown password, then you'll always get a *Label not found* error message. There is no way to trap for this inside a batch file. Yet the results may be considered desirable; after all, the user doesn't get to "break into" the program.

The Hidden Solution

The problem with the Goto command solution is that the user can still type out the batch-file program to see what the labels are. The best solution would be a batch file that could monitor something the user couldn't see. This is now possible, thanks to DOS 5.0 and its ability to hide files. (If you don't have DOS 5.0, or have yet to upgrade, then feel free to skim through the following information. If you use a third-party utility to hide files, the batch file mentioned near the end of the chapter will function properly.)

The Attrib command

The Attrib command included with DOS 5.0 can hide files—even subdirectories—on disk. It can also be used to display those hidden files, a feat that the Dir command cannot accomplish. (If you want to get picky, it's still possible for an astute DOS user to locate a hidden file, but Attrib does protect you from the casual user's prying eyes.)

Attrib was mentioned briefly at the beginning of Chapter 6, where you saw how to use *attrib +R filename* to make a file read-only so you can't accidentally delete the file. The same format of the Attrib command can be

used to make a file invisible, but the *+H* (hidden) switch is used. This extra level of hocus-pocus, added with DOS version 5.0, allows you to hide just about anything on disk. A hidden file cannot be seen in a directory listing. It cannot be copied with the Copy command or deleted with *del *.**, which gives such a file extra security over just being invisible.

Of course, hiding files and subdirectories around your hard drive has its drawbacks: Just as when you hide things in real life, you often forget where they're hidden. Imagine the panic you would feel if Grandma told you that old book you just threw away had $100 bills pressed between the pages! Fortunately, the Attrib command can locate hidden files as well.

With DOS versions prior to 5.0, the Attrib command could be used with a wildcard to display the read-only and archive attributes of all matching files. With DOS version 5.0, you can now see which files have system attributes as well as hidden attributes. For example, switch to the root directory of drive C by entering the following:

```
C>cd \
```

Now use the Attrib command to display the attributes of all the files in your root directory. Enter:

```
C>attrib *.*
     SH        C:\IO.SYS
     SH        C:\MSDOS.SYS
  A            C:\AUTOEXEC.BAT
  A            C:\COMMAND.COM
      R        C:\WINA20.386
  A            C:\CONFIG.SYS
```

What you see on your screen will be different from what's shown above, but you'll definitely have some hidden and system files there, denoted by the H and S attributes before the file name. Above, IO.SYS and MSDOS.SYS are hidden files with their system attribute set. The A, or Archive, attribute denotes a file that has been modified since the last backup. And the R attribute, before WINA20.386 above, indicates a read-only file.

To make any file invisible, you use the Attrib command in the following format:

attrib *+H filename*

+H is the switch that turns on the file's hidden attribute, rendering it invisible. Conversely, if you use a minus sign in front of H, as in *–H*, you make a hidden file visible.

filename is the name of the file, or of a group of files specified with a wild-card, that you want to make invisible.

When you make a file invisible, note that you don't erase it. True, you won't see it anymore (unless you use *attrib* *.*), but it's still there. Making a program file invisible doesn't make the file useless; you can still run the program by typing in its name. However, you cannot use the Del, Copy, or Ren (Rename) commands on an invisible file.

One last password batch file effort

A final way to run a password batch file is to use the If command with its *exist* option. *If exist* is used to test for the existence of a file. If the file exists, the condition is true and the rest of the If command is executed. Using this test, you can create a file with a specific name and use that file as your password. Add the power of Attrib's file-hiding ability and you have a level of secrecy, because it's not obvious which file in the current directory is the password (unless the user lists the batch file, in which case the whole exercise becomes futile—but you knew that already).

The following is PASSWD3.BAT, a final stab at showing how passwords can be included in a batch file. Here the password typed after *passwd3* must equal the name of a file on disk. The immediate drawback? In the batch file in Figure 7-13, any file on disk will pass the test. But it's an interesting example.

```
1:     @echo off
2:     if "%1"=="" goto ERROR
3:     if exist %1 goto PASSED
4:     echo Invalid password
5:     goto END
6:   :ERROR
7:     echo Type your password after passwd and try again
8:     goto END
9:   :PASSED
10:    echo You're in!
11:    goto end
12:  :END
```

Figure 7-13. *PASSWD3.BAT uses the If command with* exist.

Enter this program using your text editor or word processor. Save it on disk using the name PASSWD3.BAT.

The structure and flow of this password batch file is similar to the other two discussed earlier in this chapter. The main difference here is line 3, the If command's *exist* test. *if exist %1* tests to see whether the password

matches a file somewhere on disk (since the password could also be a full path name). If so, the program proceeds down to the label PASSED at line 9 and executes whatever sensitive programs would be listed there. Again, the fault to this method is that typing any file name will pass the test. However, consider the program PASSWD4.BAT in Figure 7-14.

```
1:     @echo off
2:     if "%1"=="" goto ERROR
3:     if exist %1 goto %1
4:     echo Invalid password
5:     goto END
6:   :ERROR
7:     echo Type your password after passwd and try again
8:     goto END
9:   :SECRET
10:    echo You're in!
11:    goto end
12:  :END
```

Figure 7-14. *PASSWD4.BAT, which uses If* exist *and Goto.*

This version solves two problems: It avoids the accidental *Label not found* error by confirming that a file exists in line 3. Also, it relies upon a specific file name, SECRET, which is the label at line 9.

Copy the file PASSWD3.BAT to a new file named PASSWD4.BAT. Then use Editor, Edlin, or a word processor to enter the changes noted in Figure 7-14 (in italics). Save the file, then enter the following at the DOS prompt:

```
C>passwd4 secret
Invalid password
```

Huh? But *secret* is the password! While this is correct, the problem is that you need a file on disk named SECRET. (In the unlikely event that you had a file on disk named SECRET, the program would have worked.) This will foil but not completely fool some of those who type out batch files. But to get things to work, you should create a file named SECRET somewhere on your disk, such as the current directory. (A path name entered here wouldn't match the SECRET label. Therefore it must be a file in the current directory.)

To create a secret file, enter the following:

```
C>type nul > secret
```

This is a little trick that creates a zero-length file. When you use the Type command with the NUL device, nothing is generated. The output, nothing, is then redirected to a file. Because DOS has to initially create the file SECRET for redirection, nothing ends up in that file. To prove it, enter:

```
C>dir secret

 Volume in drive C is FORSYTHWOLF
 Volume Serial Number is 1810-1496
 Directory of C:\

SECRET              0 10-19-92  11:18a
        1 file(s)           0 bytes
                    2678784 bytes free
```

Now you can use the Attrib command to make SECRET invisible. Enter the following:

```
C>attrib +h secret
```

Now the file is hidden and ready for use with the password batch file. Enter the following:

```
C>passwd4 secret
You're in!
```

Remember that this as well as any other attempt at secrecy in a batch file is really futile. Even hiding files, programs, batch files, and subdirectories on disk is only a half solution to security; anyone knowledgeable about DOS can undo whatever you've done. Third-party utilities and solutions exist. Look them up if your data is truly sensitive and requires adequate password protection.

Chapter Summary

- The program REPLY.COM lets you write interactive batch commands that present menus and evaluate a user's response to the menu choices.
- A batch file can display help information—or any other additional information—at the press of a key.
- A batch file can erase or change part or all of the information it displays.
- Chapter 15 describes a menu system that will help you use your computer more efficiently.

- There are ways to add security to a batch file, such as using the program REPLY.COM with a secret key code.
- The program ASK.COM is used like REPLY.COM but scans only for a Y or N keypress. This allows you to skip over dangerous or sensitive parts of a batch file more easily than using REPLY.COM alone.
- Although batch files can impose some password security, they really don't offer solid security against anyone who can type out a batch file's contents.
- The Attrib command can be used to make certain files invisible or hidden. It can also display hidden files and unhide or change the attribute of any file or subdirectory.

SECTION

II

Customizing Your System

CHAPTER

8

Controlling the Environment and CONFIG.SYS

DOS sets aside a small area of memory to keep track of certain information, such as the names of the directories in the command path and the definition of the system prompt. This area of memory is called the *environment;* you can examine and change it with the Set command.

The term *configuration* refers to the hardware DOS must control and the way DOS allocates (or *configures*) the computer's memory. A file named CONFIG.SYS in the root directory of the system disk contains configuration commands that give DOS instructions about its configuration.

This chapter shows you how to control the environment and CONFIG.SYS to your advantage.

The Environment

The environment contains *environment variables,* each of which has a name and value. The Set command displays, changes, or deletes these environment variables. For example, enter a Set command with no parameters to display the name and value of each environment variable:

```
C>set
```

You should see something like this:

```
COMSPEC=C:\DOS\COMMAND.COM
PATH=C:\DOS;C:\BATCH;C:\WORD;C:\123;C:\
PROMPT=$n$g
TEMP=C:\TEMP
C>_
```

The name of each environment variable is to the left of the equal sign, and the value is to the right:

Name	*Value*
COMSPEC	C:\DOS\COMMAND.COM
PATH	C:\DOS;C:\BATCH;C:\WORD;C:\123;C:\
PROMPT	ng
TEMP	C:\TEMP

The COMSPEC (for *command specification*) environment variable tells DOS where the command interpreter is located—the drive letter and path name of the program that interprets the commands you enter—and passes instructions along to DOS. Usually, this program is COMMAND.COM, as shown in the example above. You needn't concern yourself with the COMSPEC environment variable, except to make sure that you don't erase or move COMMAND.COM from the specified drive and directory.

The PATH environment variable shows your search path. Listed is every directory you specified in the most recent Path command. If you haven't specified a command path, there is no PATH environment variable.

The PROMPT environment variable shows the system-prompt definition you specified in the most recent Prompt command. If you haven't defined a system prompt, there is no PROMPT environment variable.

The TEMP environment variable tells DOS where to store temporary files. This is used by some DOS commands, such as MORE and SORT. Some programs, such as Windows, use the TEMP variable to identify a subdirectory in which they can store temporary files. (Temporary files are usually deleted before the program that created them returns to DOS.)

The environment lets DOS keep track of this information regardless of the current drive or directory or what program is running. You can use the Set command to display or change these environment variables or to create new ones.

Controlling the Environment with the Set Command

A moment ago you entered a Set command with no parameters to display the environment variables. To change an environment variable, you include a parameter that consists of the name of the variable, an equal sign, and the new value:

set variable=*new value*

For example, to change the system prompt, you would type *set prompt=* followed by the new prompt definition.

Because you may have a long prompt already defined, you don't need to type this example, but if you entered *set prompt=[$p]*, the system prompt would be the current drive and directory enclosed in brackets. Displaying the environment variables again with the Set command would show *PROMPT=[$p]*. Typing *set prompt=[$p]* has the same effect as typing *prompt [$p]*; you can change the PATH and PROMPT environment variables with either the Set command or the Path and Prompt commands. (This is not true of other environment variables.)

You can delete an environment variable by typing *set* followed by the name of the environment variable and an equal sign. For example, entering *set prompt=* (with nothing after the equal sign) has the same effect as entering *prompt* by itself: It deletes any system-prompt definition and reverts to the standard system prompt (C>, for example).

Using Environment Variables in a Batch File

You can use the name of an environment variable in a batch file to represent the variable's value, much as you would use a replaceable parameter. When DOS finds the name of an environment variable enclosed in percent signs (such as *%prompt%*) in a batch file, it replaces the name with the value of the environment variable, just as it replaces *%1* with the first parameter typed with the batch command.

For example, in Chapter 3 you may have created a batch file to restore your preferred system prompt. Here's a variation on that short batch file that uses an environment variable to display the current prompt definition. As a side effect, by using I/O redirection, this batch file also creates a new batch file named PROMPTRS.BAT that defines the current prompt. Enter the following to create ENV.BAT:

```
C>copy con env.bat
@echo off
echo The current prompt definition is %prompt%
echo prompt %prompt% > promptrs.bat
^Z
```

Now run ENV.BAT:

```
C>env
```

DOS should display *The current prompt definition is* followed by the definition of the system prompt. The output of the second Echo command—*prompt* followed by the current prompt definition—is redirected to create PROMPTRS.BAT.

> ❑ **NOTE:** *If you use the default DOS prompt, there is no environment variable named PROMPT; the environment variable* %prompt% *in each Echo command is replaced with nothing (not a space, but no character at all). Nothing, therefore, follows the message* The current prompt definition is, *and PROMPTRS.BAT contains simply* prompt *(which, of course, defines the default prompt).*

To verify that PROMPTRS.BAT works, type the following to change the system prompt to COMMAND:

```
C>prompt COMMAND:<Space><Enter>
```

The system prompt changes to COMMAND:. If you display the environment variables by entering *set*, they should include PROMPT=COMMAND:.

Now use the batch command created by ENV.BAT:

```
COMMAND: promptrs
```

DOS should echo the word *prompt*, followed by the previous prompt definition, and the system prompt should return to its previous form. You can confirm that the PROMPT environment variable has changed by entering *set* once again.

If you run ENV.BAT when your preferred system prompt is defined, you create PROMPTRS.BAT, which restores your preferred system prompt.

Displaying the Directories in the Command Path

The For command has some interesting quirks, or features, one of which lends itself well to the PATH environment variable. According to the manual, the For command performs a specific DOS operation on a group of files. For example, entering *for %a in (autoexec.bat config.sys) do type %a* is a handy way to quickly list the contents of both your AUTOEXEC.BAT and CONFIG.SYS files.

Note that the items between parentheses in a For command are traditionally separated by spaces. However, this isn't a hard-and-fast rule. You can use commas or semicolons to separate the items just as you use a space. Switch to the root directory on drive C and enter the following:

```
C>for %a in (autoexec.bat,config.sys) do type %a
```

The For command caused the contents of both AUTOEXEC.BAT and CONFIG.SYS to be displayed, even though there is a comma instead of a space between the names of the two files above. The same thing can be done with a semicolon. Enter the following:

```
C>for %a in (autoexec.bat;config.sys) do type %a
```

Again, both AUTOEXEC.BAT and CONFIG.SYS are displayed.

The interesting side effect of this is that you can use the For command to sift through the different subdirectories listed in the PATH environment variable. If you look at the typical PATH variable's contents, you'll notice that each subdirectory is separated by a semicolon.

SHOWPATH.BAT is a batch file that displays each directory in the search path by using a For command whose parenthetical set is *%path%*. The effect is the same as specifying a set of file names or wildcard characters: DOS retrieves each part of the set in turn.

If your search path includes a lot of directories, SHOWPATH.BAT is a quick way to check which directories it contains and their sequence.

Enter the following to create SHOWPATH.BAT:

```
C>copy con showpath.bat
@echo off
echo COMMAND PATH DIRECTORIES
for %%p in (%path%) do echo %%p
^Z
```

Test it by entering:

```
C>showpath
```

DOS should display something like this:

```
COMMAND PATH DIRECTORIES
C:\DOS
C:\BATCH
C:\WORD
C:\123
C:\

C>_
```

The For command fetched the names of all directories in the search path and displayed them individually. It's much easier to find a directory name in this display than to pick it out of a string of directory names separated by semicolons.

Creating Your Own Environment Variables

Environment variables give you another way of controlling the behavior of batch-file programs. For example, enter the following two Set commands to create an environment variable named DRIVE and display the result:

```
C>set drive=c:
C>set
```

Now the display of environment variables includes one that wasn't there before:

```
DRIVE=C:
```

Note that DOS makes the name of an environment variable uppercase, even if you type it in lowercase.

If you ran a batch file now that contained *%drive%* on a line by itself, DOS would replace *%drive%* with *c:*, the command to set the current drive to C. By changing the value of the environment variable named *drive* before using this batch command, you could control the current drive it sets.

You delete an environment variable that you create just as you delete any other environment variable. Enter the following two Set commands to remove DRIVE and display the result:

```
C>set drive=
C>set
```

DRIVE should be gone.

You can also create an environment variable that has multiple values, such as the PATH variable. For example, if you entered *set users=matt;mark;luke;john*, then entered *set*, the last environment variable displayed would be *USERS=matt;mark;luke;john.*

Some application programs offer certain environment variables that you can set before you run those programs. An example is the TEMP variable used with Windows and now used with DOS as of version 5.

An Environmental Concern: The Limits to Growth

Like any other environment, the DOS environment is finite. Unless you tell it otherwise, DOS initially sets aside 160 bytes for the environment. That's all the room you have for your search path, system prompt, and other environment variables created with the Set command. If you add any more and there's no room, DOS displays an *Out of environment space* error message, and the last item you attempted to stick into the environment is truncated (cropped to fit as much of it as possible into the environment).

DOS can expand the environment up to a theoretical maximum size of 32K bytes, but there's a catch: You must do so in your CONFIG.SYS file, and the change has effect only after you reset your computer. Although

you probably will never need a 32K environment, something of a more reasonable size, say 1 to 2K, is appropriate if you often see the *Out of environment space* error message. The configuration command that lets you change your environment space is described later in this chapter in the section "Increasing the Size of the Environment."

Configuration Commands

CONFIG.SYS is a file in the root directory of the system disk that contains configuration commands. Those commands tell DOS how to set up your system, hence the name CONFIG.SYS. Each time you start or restart the system, DOS reads CONFIG.SYS and carries out the commands it contains. (AUTOEXEC.BAT is run later and deals more with DOS setup than system configuration.) To change the configuration, therefore, you must change CONFIG.SYS and restart DOS.

The configuration commands tell DOS such things as:

- Which file contains a *device driver*—a program that tells DOS how to control a device such as a RAM disk or a mouse
- How much memory to set aside to handle files
- How often to check to see if you have pressed Ctrl-C (or Ctrl-Break)
- How large to make the environment
- How to deal with extra memory in your PC

You'll change CONFIG.SYS to add a new device, to try to improve disk performance, to change the size of the environment, or to let DOS use any extra memory in the computer. Remember, though, CONFIG.SYS must be in the root directory, and changes to CONFIG.SYS don't take effect until you restart DOS.

The Device configuration command

Your CONFIG.SYS file might well include some Device commands, including one or more of the following:

- device=c:\dos\setver.exe
- device=c:\dos\ansi.sys
- device=c:\dos\ramdrive.sys
- device=c:\mouse.sys
- device=c:\dos\himem.sys
- device=c:\dos\smartdrive.sys 2048 512

SETVER.EXE is included with DOS 5 and is used to enable applications designed for earlier versions of DOS to run with DOS 5. ANSI.SYS and RAMDRIVE.SYS (VDISK.SYS in PC-DOS 5) are included with DOS. Chapter 3 describes how to use the ANSI.SYS commands to control the display and keyboard, and Chapter 10 describes how to use RAMDRIVE.SYS or VDISK.SYS to create a RAM disk. MOUSE.SYS tells DOS how to use a mouse. HIMEM.SYS is a program that gives some PCs about 64K of extra RAM, which DOS 5 can take direct advantage of. HIMEM.SYS also manages the use of extended memory by applications. (This is covered later in this chapter in the section "Memory Configuration Commands.") SMARTDRIVE.SYS is a disk-caching program (discussed in the section below) and is included with DOS 5.

If you add a device that requires a device driver, such as a mouse, a memory board, or a CD-ROM drive, follow the instructions in its documentation to add the proper Device command to CONFIG.SYS.

The Buffers configuration command

When a program asks DOS to read data from a disk, DOS reads from the disk, stores the data in an area of memory called a *buffer*, and then transfers the data to the program. The Buffers configuration command tells DOS how many buffers to create in memory.

The number of buffers can affect the speed of disk operations. Regardless of the amount of data a program requests, DOS reads and stores 512 bytes (one sector) in a buffer. The next time the program asks for data from the disk, DOS first looks in its buffers. If the data is in a buffer, DOS transfers the data to the program without reading from the disk; only if the data isn't in a buffer does DOS read from the disk again. Transferring data from one area of memory (the buffer) to another (the requesting program) is much faster than reading from a disk, so buffers can save time.

Starting with version 4.0, DOS can read data from and write data to the sectors that follow the sector currently being used. This is referred to as a *secondary cache,* and it speeds up the operations of some language compilers and database programs. Note that if you're already using a disk-caching program, such as SMARTdrive (included with DOS 5), then specifying a secondary cache with buffers isn't necessary.

If all its buffers are full, DOS reuses the buffer least recently used, so the more buffers DOS uses, the more likely it is that DOS will find the data a program wants in a buffer. Up to a point, increasing the number of buffers speeds up many programs, especially database programs and DOS itself, if your directory structure has several levels.

Through version 3.3, the Buffers command has only one parameter: the number of buffers to set aside in memory. To specify 15 buffers, for example, you would put the command *buffers=15* in CONFIG.SYS.

As of version 4.0, the Buffers command has two parameters:

buffers=*number*,*advance*

number is the number of buffers to set aside. It can be from 1 through 99 in versions 3.2 and earlier, 2 through 255 in version 3.3, and 1 through 99 in versions 4.0 and later.

If you don't specify *number*, DOS assumes one of the following values based on your system hardware:

If You Have:	***DOS Sets Buffers To:***
Less than 128K RAM and 360K disk	2
Less than 128K RAM and disk larger than 360K	3
128–255K RAM	5
256–511K RAM	10
512K or more RAM	15

advance is the number of sectors that DOS will store in the secondary cache. If you're already using a caching program, such as SMARTdrive, specifying a value for *advance* isn't necessary. Otherwise, a value of 1 or 8 can be used.

As always, there's a price: Each regular disk buffer (specified with *number*) takes 532 bytes of memory, and each look-ahead buffer (specified with *advance*) takes up memory as well, reducing the amount of memory available to other programs.

Assigning too much memory to buffers slows down some programs and can even prevent a program from running at all. Creating too many buffers can slow the system down even if the program that's running isn't affected, because DOS must search through all buffers for the requested data, which sometimes can take longer than simply reading the data from the disk in the first place.

The number of buffers that DOS assumes is the minimum number needed for efficient operations. You can speed disk operations—sometimes significantly—by tailoring the number of buffers, the types of disk drives you have, the amount of memory, and the types of programs you use. It may take a bit of tinkering to find the best values. Figure 8-1 lists a value for the Buffers command that you can use as a starting point, depending on the type of system you use.

Computer Model	*Buffers=*
IBM PC and compatibles without a fixed disk	8
IBM PC/XT and compatibles with a fixed disk, IBM PC/AT, PS/2, and compatibles without a fixed disk	16
IBM PC/AT, PS/2, and compatibles with a fixed disk	32

Figure 8-1. *Suggested values for the Buffers configuration command.*

Add the Buffers command to CONFIG.SYS, restart DOS, and use your system awhile. If you're satisfied with the improvement, you're done; if you're not satisfied, increase the number of buffers by half (substitute 12, 24, and 48 for the preceding numbers), restart DOS, and try your system again. If the increased number of buffers makes the system faster, you should probably be satisfied; more buffers could slow things down. If the increased number of buffers makes the system slower, reduce the number of buffers and try again. You can probably find the number of buffers that results in faster system operation than the number that DOS assumes.

The Files and FCBS configuration commands

Just as you must open a file folder before you can use the papers it contains, DOS must open a disk file before it can use the data it contains. DOS limits how many files a program can have open at one time; the Files and FCBS configuration commands let you control this limit.

If you don't specify otherwise, DOS assumes the value 8 for the Files configuration command and 4 for the FCBS command. This should be sufficient for most routine work. Some programs, however, such as database programs and local area networks, may require that you increase one or both of these limits, especially if you also use the Share command, which permits file sharing for networks and multitasking programs. The program's manual should tell you what values to specify with the Files and FCBS configuration commands that you put in CONFIG.SYS.

It's generally a good idea to set a value for Files but to ignore the FCBS command unless you're running older software that requires it. (The Files command actually replaced FCBS in an earlier version of DOS; FCBS was kept around to maintain compatibility.) Files can be set to any value from 1 through 255. A good number of files to have is 20, which you achieve by putting the line *files=20* into your CONFIG.SYS file.

Raising the limit on open files increases the amount of memory that DOS uses by 48 bytes for each additional file.

The Break configuration command

The Break configuration command controls how quickly DOS responds when you press Ctrl-C (or Ctrl-Break). DOS normally checks to see whether you have pressed Ctrl-C only when it reads from or writes to a disk drive, printer, serial port, or other device. However, if you include *break=on* in CONFIG.SYS, DOS responds to Ctrl-C as soon as you press it.

You can also control *break* with the DOS Break command. Entering *break on* has the same effect as including *break=on* in CONFIG.SYS. Entering *break off* cancels the effect. Some application programs turn *break* off and leave it off when they return to DOS. To check *break* status, enter *break*; DOS responds with either *BREAK is on* or *BREAK is off*.

If you're working on batch files or programs, or if you're using programs that spend much of their time in computations that don't require a device, setting *break=on* sometimes lets you avoid having to restart DOS.

Increasing the Size of the Environment

The program that communicates with you, displaying prompts and other messages and interpreting the commands that you type, is called a *shell* because it surrounds DOS; when you look at the screen, you see the shell, not the inner workings of DOS. The shell program that almost all DOS computers use is called COMMAND.COM. The Shell configuration command lets you name the shell program to be used; if you don't specify one, DOS assumes you want to use COMMAND.COM.

Starting with version 4.0, DOS includes a program called DOSShell that lets you manage files and start programs by choosing items from a menu rather than by typing commands. This is really a shell for a shell; DOSShell is a shell for COMMAND.COM, which is itself a shell for DOS. Note that you don't use the Shell configuration command to name the DOSShell program.

Most DOS users don't need to name a shell program, but the Shell command has an additional interesting and valuable capability: If you use it to name COMMAND.COM—a redundant command because DOS almost certainly would use COMMAND.COM anyway—you can use a parameter that increases the size of the environment. This removes a potential restriction on the length of your command path or prompt, and can give you room to define several of your own environment variables.

To increase the size of the environment, put the following Shell configuration command in CONFIG.SYS:

shell=c:\dos\command.com /p /e:*size*

The parameter *c:\dos\command.com* gives the path and name of the shell program.

/p specifies that the shell program is to be permanent. Be sure to include */p*; this ensures that DOS will work just as it always does and that your AUTOEXEC.BAT file will run (if present in the root directory).

/e:size specifies the size of the environment. *size* is the size of the environment in bytes, from 160 through 32,768. Note that DOS will round whatever value you enter to the nearest multiple of 16. Unless memory is scarce, specify at least 512 bytes.

Suppose you wanted to set the environment to 800 bytes, and that COMMAND.COM is located in the root directory of your C drive. You would put *shell=c:\command.com /p /e:800* in CONFIG.SYS.

Memory Configuration Commands

With the introduction of version 5, DOS now offers memory configuration options for CONFIG.SYS. The idea is to give you more memory in which to run DOS programs, which these options do by moving DOS commands and device drivers out of main memory. The subject of DOS and memory, and additional information on how these commands work, is covered in the next chapter. The following information is specific to those memory configuration commands used in CONFIG.SYS.

Memory device drivers

The good news about DOS and memory is that solutions exist to give you more memory under DOS and to take advantage of that memory. Prior to DOS 5, these solutions were available in third-party software. But now these solutions are part of DOS itself. The bad news is these tricks are effective only on computers with an 80286, 80386, or 80486 microprocessors. The PC/XT crowd cannot take advantage of these configuration commands.

In DOS 5, the Device configuration command is used to load one or two memory device drivers. These device drivers must be installed first for you to take advantage of the other memory configuration commands. The two device drivers are HIMEM.SYS and EMM386.EXE.

Controlling extended memory with HIMEM.SYS

The HIMEM.SYS device driver is used on 80286, 80386, and 80486 systems with at least 256K of extended memory. (The subject of extended memory is covered in detail in Chapter 9.) It converts the first 64K of that memory

for use by DOS as a *High Memory Area*, or HMA. It also provides basic memory-management support for all your extended memory. HIMEM.SYS is what's known as an Extended Memory Specification (XMS) memory manager.

HIMEM.SYS is installed using the Device command. Putting the line *device=himem.sys* in your CONFIG.SYS file sets your computer up to use extended memory under DOS. Remember to specify a full path, such as *c:\dos\himem.sys*, if HIMEM.SYS isn't in your root directory.

Several commercial applications take advantage of HIMEM.SYS and the extra memory it gives DOS. One of the first was Microsoft Windows, which uses HIMEM.SYS to add an extra 64K of RAM to the basic 640K in most DOS computers. With DOS 5, however, HIMEM.SYS gives you power that DOS itself can tap directly: You can use HIMEM.SYS and the Dos configuration command to move all of DOS's basic files into extended memory. This means on some systems you can have almost all of the 640K of conventional memory available to your programs.

The format of the Dos command is:

dos=*high/low,umb/noumb*

high/low tells DOS whether or not it can be loaded into the HMA setup by HIMEM.SYS. When *high* is specified, DOS loads itself and all its files into HMA, freeing up quite a bit of your 640K conventional memory. When *low* is specified, DOS loads itself into the 640K conventional memory, just as it did in all previous versions of DOS.

umb/noumb controls what are called *upper memory blocks*. These are areas of unused memory between the 640K mark and the top of DOS memory, 1024K. When *umb* is specified, DOS maintains a link between conventional memory and the upper memory blocks; when *noumb* is specified, DOS doesn't maintain links, just as in previous versions.

Say you're using desktop publishing, CAD, or graphics applications and you need extra RAM. Adding the following two lines to your CONFIG.SYS file would give you more room under DOS 5:

```
device=c:\dos\himem.sys
dos=high
```

Setting Dos to *high* places all DOS's files into the HMA. On some systems, this has the result of freeing up almost all 640K of conventional memory—providing you have enough extended memory installed.

Remember that you need to put the line *device=himem.sys* in your CONFIG.SYS file to use the Dos command. (The position is important because the line must precede any Device command specifying a device driver that uses extended memory.)

Using expanded memory with EMM386.EXE

The EMM386.EXE driver is used to convert extended memory into expanded memory. Note that the *EXE* extension is not a typo; this is a rare device driver that doesn't have the SYS extension. (EMM386.EXE is also a program you can run at the DOS prompt. It's dual-purpose.)

EMM386.EXE is useful, since many DOS applications that use extra memory in a PC use expanded and not extended memory. (The differences between extended and expanded memory, and all the other memory terms, are covered in the next chapter.)

As its name implies, EMM386.EXE is used only on DOS computers with an 80386 or 80486 microprocessor. The device driver takes advantage of the special memory-mapping power built into those chips. If you have an 80286, you can buy third-party device drivers that do similar things, such as MOVE'EM from Qualitas or QRAM from Quarterdeck Office Systems.

To install the EMM386.EXE device driver, you use the device command in your CONFIG.SYS file. Two options follow EMM386.EXE, in the following format:

device=c:\dos\emm386.exe *memory options*

memory is the amount of extended memory you want EMM386.EXE to convert into expanded memory. The value ranges from 16, for 16K, through as much extended memory as you have in your system, with a theoretical maximum of 32,768K. If *memory* isn't specified, 256K is assumed.

options represents a slew of optional switches and parameters that customize the way the EMM386.EXE driver controls memory. These options, however, are not needed under most circumstances. Only if you experience problems or some application makes suggestions for how to set up EMM386.EXE should you use the options.

Once you have EMM386.EXE installed, you can use two other commands to help free memory. These commands will move some DOS options, memory buffers, and memory-resident programs out of main memory and into available upper memory blocks. The two commands that do this are Devicehigh, used in your CONFIG.SYS file, and Loadhigh, which is used

primarily in AUTOEXEC.BAT. Note that you must specify the *umb* option with the Dos command in your CONFIG.SYS file before you use either of these commands.

The Devicehigh command is used to load device drivers into the upper memory area. Normally, device drivers are installed into your 640K of conventional memory, taking up memory you could otherwise use for running programs. But by using Devicehigh in CONFIG.SYS, you can load device drivers elsewhere, freeing memory for applications.

The Loadhigh command works exactly like Devicehigh, but it loads memory-resident programs into the upper memory area. Normally, these memory-resident programs would sit in your 640K of conventional memory.

If you want to run DOS with as little memory overhead as possible, the following lines should be part of your CONFIG.SYS file:

```
device=c:\dos\himem.sys
dos=high,umb
device=c:\dos\emm386.exe 1024
```

The first line loads the HIMEM.SYS driver, which sets up the computer with extended memory management and defines the HMA. The line *dos=high,umb* loads DOS high (into the HMA), freeing your conventional memory and establishing the UMB links for later use with the Devicehigh and Loadhigh commands. And finally, *device=c:\dos\emm386.exe 1024* converts one megabyte (1024K) of extended memory into expanded memory for programs that use expanded memory.

Notice that *device=c:\dos\emm386.exe* should come *after* the line *device=himem.sys* in your CONFIG.SYS file. Also, you should remove any other expanded memory device drivers in your CONFIG.SYS file (such as 386MAX or QEMM) if you plan on using EMM386.EXE. Finally, if you're using Windows 3.0 exclusively, you do not need to specify EMM386.EXE in your CONFIG.SYS file. Windows uses its own built-in extended-to-expanded memory conversion.

CHAPTER

9

Getting the Most from Memory

Ever since the earliest days of microcomputers (and larger computers even before them), limited memory storage has been an issue. The first personal computers used 8-bit microprocessors that could access a massive 64K of memory storage, yet because of the steep price of memory chips back then, most systems came with only 4K of memory—you made everyone jealous if you had 16K.

When the IBM PC was introduced, it came with 64K already installed—but with a potential for up to 640K of memory. That amount was considered ludicrously large. In a world where all work on microcomputers was getting done comfortably in 64K of RAM, who would need 640K? Eventually, everyone found ways to use all that memory, and only a few years after the PC became popular, 640K was no longer considered enough.

Now the big problem is the 640K limit. In a computer with an 80386 microprocessor, capable of accessing billions of bytes of memory, you can still use only 640K for DOS and its applications. Since DOS is so popular, that creates a dilemma: How can you use more memory on a powerful computer that's running a 10-year-old operating system—without waving goodbye to your beloved DOS software?

Thanks to today's advanced technology, you can install a lot of memory in your PC. But thanks to yesterday's operating system, DOS, you can't use all that memory. One option is to use a different operating system, such as OS/2. But DOS users own DOS programs, which run only under DOS. This quandary has several solutions, a few of which are available within DOS itself, starting with version 5.0.

The PC and Memory

All computers running DOS today run like the first IBM PC, with its (now) limited 640K of memory. The reason for this is *compatibility*. DOS had to remain compatible with the millions of existing 8088 PCs when the 80286 came along. Yet the 80286 in the PC/AT was able to access 16 times as much memory as the original PC. To keep DOS compatible with all PCs, the extra memory in 80286 computers often went unused except for such memory tools as RAM disks and disk-caching programs. (These subjects are covered in Chapter 10.)

Later the 80386 and 80486 came along, each of which could access much more memory than the sole megabyte available in the first PC. Still, these advanced chips are limited to using memory just like that first PC when they run DOS programs. The total amount of memory available to each microprocessor is shown in Figure 9-1.

Microprocessor	*Maximum Usable Memory*
8088/8086	1 megabyte
80286	16 megabytes
80386	4 gigabytes (4096 megabytes)
80486	4 gigabytes (4096 megabytes)

Figure 9-1. *Microprocessors and the full amount of memory they can support.*

Of course, most people never fully populate their computers with all the potential RAM. In fact, the typical 80386 computer may have only 4 megabytes of RAM installed, and the system's design may further limit it to a physical maximum of 16 megabytes. But this doesn't rule out the chip's potential to access 4 gigabytes—which is beyond DOS's reach.

At this point some cheery individual usually raises his or her hand and asks, "Why not just fix DOS to use all that memory directly?" The answer is, they did: It's called OS/2. But although OS/2 can use all that memory, it cannot run the wealth of DOS applications.

Fortunately, solutions are at hand to put all that memory to immediate use (although it may not be used as effectively or as efficiently as it would be under an advanced operating system such as OS/2). Before DOS 5.0, these solutions were available only as third-party utilities. But starting with DOS 5.0, these memory-enhancing techniques are now part of DOS itself.

How the PC Uses Memory, and Terms to Describe That Memory

IBM was pleasantly surprised by the PC's great success. Had it been anticipated, IBM might have given more forethought to future designs of the machine around new microprocessors. Alas, we're left with a PC that has several unusual, often funky, terms to describe how its memory is used. If you want to take advantage of the memory solutions now offered, a good understanding of these terms is in order.

RAM and ROM

All memory inside a computer is available for use by the microprocessor. The first PC could access only 1 megabyte of memory, which was divided into two types: RAM and ROM.

RAM stands for *random access memory*. DOS runs in RAM, your programs load into RAM, and all data is created and manipulated in RAM. A microprocessor needs RAM to operate, and the more RAM a computer has, the more it can do.

ROM stands for *read-only memory*. ROM is accessed just like RAM; the microprocessor can't really tell the difference. But ROM is *read only*: Its contents cannot be changed or erased, and they don't disappear when the power is turned off (unlike RAM, which goes *poof* when you turn off your computer).

ROM is where the PC stores its *BIOS*, which stands for the *basic input/output system*. The BIOS contains the basic instructions for your PC, including primitive directions for using the screen, keyboard, and other I/O devices. Other ROMs in your PC reside in the hard-drive controller, the EGA or VGA adapter, and any network card.

The 640K barrier

When IBM designed the first PC, it split the 8088's 1 megabyte of memory into two parts: one for RAM and the other for ROM. This gave the computer a whopping 640K in which to run programs and another 384K for the BIOS and "future expansion."

The 640K of program RAM in a PC is referred to as *conventional memory*. The upper 384K, reserved for future expansion and ROM, is called *reserved memory*, or the *upper memory area*. You'll see how these terms fit into the big picture later in this chapter.

The number 640K has been a thorn in the side of PC users for over a decade; that's all the RAM a PC has in which to run programs. And that area, although 10 times bigger than the memory available to older microcomputers, just isn't enough. When you consider that some versions of DOS occupied about 90K of that 640K, not much memory is left to run programs. As programs became larger, 640K became a frustrating barrier. Confounding this problem is the fact that the 80286 and later microprocessors can access gobs more memory but are blocked by the 640K limit.

Expanded memory

The first users to bump their heads against the 640K barrier were dedicated spreadsheet users. Spreadsheets use an incredible amount of RAM, especially as both the programs themselves and users' worksheets grow. A solution was devised to give the PC an extra pool of memory from which spreadsheets could draw more RAM: the *LIM expanded memory standard,* or EMS. LIM stands for Lotus (the spreadsheet maker), Intel (the microprocessor maker) and Microsoft (the DOS maker), the three primaries behind the new standard.

Expanded memory is an extra pool of memory, up to 32MB, that a PC can use to store data. (Applications can't run in expanded memory.) Unlike extended memory, which is memory above the 1-megabyte mark, expanded

memory uses a free 64K block of memory between 640K and 1MB. That 64K serves as a window into which the expanded memory manager switches 16K blocks of expanded memory. Although you can access only 64K of expanded memory at a time (some EMS boards can increase the size of that window), it can come from anywhere within the 32MB. This process of switching blocks of memory in and out of the 64K window is called *bank switching*.

Applications that need the expanded memory use the EMS software and a memory device driver (loaded by CONFIG.SYS) to access that memory. EMS isn't perfect, but it's a usable system that many DOS applications adhere to. The most recent version of EMS, LIM 4.0, has lots of potential and is supported directly by DOS 5. (Other versions of DOS can access LIM 4.0 EMS via a combination of hardware and software, covered later in this chapter.)

Extended memory

Extended memory isn't the same as expanded memory. Often the two are confused, and for good reason—their names are annoyingly similar. Basically, extended memory is any memory beyond 1 megabyte in an 80286 or later computer.

The weakness of extended memory is that DOS cannot directly use it to run programs. The microprocessor can access that memory only by running in a special *protected* mode. But DOS can't use protected mode; it's an operating system designed for the 8088, which had no protected mode. (OS/2 does use extended memory because it runs in protected mode.) So, except for some specially written programs, extended memory is often limited to use as a RAM disk, or disk cache, or for print spoolers and other memory-related programs (which are covered in the following chapter). But under DOS, extended memory usually remains underutilized.

The high memory area

In the mid-eighties, Microsoft found a way to use part of extended memory under DOS. Using a device driver called HIMEM.SYS, Microsoft was able to give Windows an extra 64K of RAM on machines that had extended memory. This first 64K of extended memory is referred to as the *high memory area,* or HMA.

Although 64K doesn't seem like much, every little bit helps. (And starting with DOS 5.0, that 64K has proven to be a boon to memory-hungry DOS users.) The secret behind that extra 64K was a particular quirk in the design of 80286 and higher microprocessors: It seems you can fool the computer into thinking it can "see" an extra 64K bank of memory right

above the top of the traditional 1-megabyte mark. With HIMEM.SYS and enough extended memory installed, a number of programs can take advantage of the HMA.

In addition to defining the HMA, the extended memory manager HIMEM.SYS also follows a standard for interfacing with extended memory. The *extended memory specification,* or XMS, provides a common way for applications to access extended memory. Although that memory still can't be used under DOS, certain other device drivers can be used under the XMS to convert extended memory into expanded memory for use with DOS, and some DOS programs (such as Windows) are smart enough to use the XMS and extended memory themselves. That's the secret to putting extended memory to work: With an XMS driver, you can access extended memory under DOS. All your system's extended memory becomes usable and can even be converted into expanded memory for use by DOS applications.

The solutions

The purpose behind all this concern with memory is to give you and your applications access to it while you run DOS. Prior to DOS 5.0, you could get access to extra memory using third-party memory device drivers, installed in CONFIG.SYS. But DOS 5.0 comes with an XMS driver, an EMS driver, and all sorts of utilities that let you take advantage of extra memory under DOS.

Figure 9-2 shows terms described in previous sections and a *memory map* to show you which terms apply to which portions of memory.

On a typical PC, DOS can use 1MB of memory; the first 640K is conventional DOS memory, where programs run, and the remaining 384K is reserved memory, or the upper memory area.

On a PC with expanded memory, the memory map looks the same. Expanded memory is not memory "above the 1MB mark." Instead, it's a pool of memory separate from the 1MB of memory DOS uses. The EMS device driver controls access to expanded memory, often through one of the memory locations in reserved, or high DOS, memory. (Expanded memory is supplied using a combination of LIM hardware and software; the actual memory is on the hardware, whereas the software provides the EMS device driver, which is accessed by EMS-aware applications.)

On 80286 or later PCs, the extra memory is usually *extended memory*. DOS can access the first 64K of extended memory using HIMEM.SYS, and the rest of extended memory using the XMS driver. But for extended memory to be usable with many DOS applications, it must be converted into expanded memory.

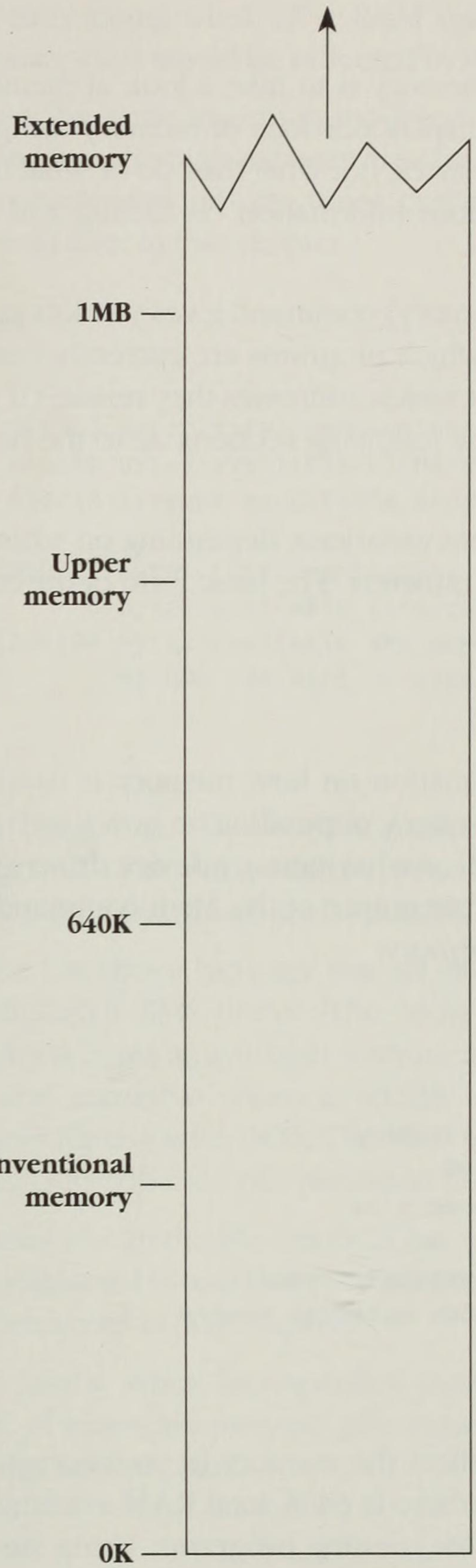


Figure 9-2. *DOS's memory map and the terms that describe different parts of memory.*

The Mem Command

A good way to see how DOS uses memory is to take a look at memory. The Debug program allows you to display portions of memory using its Dump command. But all you see are bytes, not what they do or what they represent. (Refer to Chapter 5 for more information on Debug and the Dump command.)

Starting with DOS 4, the Mem (for memory) command gives you lots of information about memory, including which programs are currently loaded into memory, how big they are, and at which addresses they reside. (If you don't yet have DOS 4 or later, skim the following sections up to the heading "Doing It Without DOS 5.")

The Mem command has three different variations, depending on whether you supply one of its two optional parameters. The basic form of the command uses no parameters, as follows:

```
mem
```

After entering *mem*, you'll see information on how memory is used on your PC. The display varies for each system, depending on how much and what kind of memory you have as well as what memory device drivers you have installed. The following shows the output of the Mem command on an 80386 system that has 2MB total memory:

```
C>mem


    655360 bytes total conventional memory
    655360 bytes available to MS-DOS
    579648 largest executable program size

   1310720 bytes total contiguous extended memory
   1310720 bytes available contiguous extended memory

C>_
```

What you see on your screen will reflect the memory in your computer. Above, the Mem command says that there is 640K total RAM available in the computer, with about 566K free for running programs. Using simple math, that means DOS, device drivers, and memory-resident programs are occupying the remaining 74K.

Also, the Mem command tells you that the computer has 1.25 megabytes of extended memory available. Since that same amount is listed as *contiguous extended memory,* you can assume it's all available, or not

currently being used. (If a RAM disk or disk cache were in effect, that memory used would be reflected in the final value.)

When the same system is modified to include an XMS driver with DOS loaded into the high memory area (under DOS version 5), the output of the Mem command reflects those changes. (Loading DOS into the HMA is covered later in this chapter.)

```
C>mem

    655360 bytes total conventional memory
    655360 bytes available to MS-DOS
    605712 largest executable program size

   1310720 bytes total contiguous extended memory
         0 bytes available contiguous extended memory
   1245184 bytes available XMS memory
           MS-DOS resident in High Memory Area

C>_
```

Note the increased amount of memory available. 605K is free for running programs. (The difference between 605K and 640K is the size of loaded device drivers, including HIMEM.SYS, and memory-resident programs.)

In the list above, you see that all extended memory is controlled by the HIMEM.SYS XMS driver. The Mem command tells you this by listing 1,245,184 bytes as available XMS memory. The difference between the *total* and *available* values is 65,536 bytes—or the 64K that HIMEM.SYS defines for use with DOS. The XMS memory is now available, but keep in mind DOS still can't run programs there.

You see also in the list that DOS has been loaded into the HMA, which was accomplished by the Dos configuration command in CONFIG.SYS. (That's covered later in this chapter.)

And finally, when an expanded memory manager is installed to convert 512K of extended memory into expanded memory, the Mem command's display reflects that:

```
C>mem

    655360 bytes total conventional memory
    655360 bytes available to MS-DOS
    597488 largest executable program size
```

```
    917504 bytes total EMS memory
    524288 bytes free EMS memory

   1310720 bytes total contiguous extended memory
         0 bytes available contiguous extended memory
    622592 bytes available XMS memory
           MS-DOS resident in High Memory Area

C>_
```

These numbers may seem a bit confusing. Assuming this is the same PC, there were originally 1,310,720 bytes of extended memory. 64K was used in the HMA, which left 1,245,184 of extended memory. In the example above, 512K of that extended memory has been converted into expanded memory by the EMM386.EXE device driver. So where did *917504 bytes total EMS memory* come from?

The missing value is 384K, which is 917,504 (896K) minus 524,288 (512K). That 384K happens to be the *reserved* area of memory, between the 640K mark and 1MB, also called the *upper memory area*.

These three examples of the Mem command cover its output for most popular configurations of the PC's memory. Using Mem, you can find out how much conventional, extended, and expanded memory is available, in addition to whether an XMS or EMS driver is being used. But so far the explanations have only told you what this means. Later in this chapter, you'll learn how to set up, control, and use that memory.

The Mem command's optional parameters

The Mem command has three optional parameters to display degrees of information about memory. Two advanced parameters are */p* or */program* and */d* or */debug*. Mortal DOS users can use the */c* or */classify* switch. (You can try */p* or */d* on your own; impress your friends with the display!)

As with the Mem command by itself, entering the *mem /c* command will display different output on each PC. Figure 9-3 shows the output of *mem /c* as it would appear on an 80386 PC with 4MB of RAM, with an XMS and EMS memory device driver installed.

In Figure 9-3, the top part of the display contains three columns that describe programs in memory. The bottom part is similar to what you'd see if you entered the Mem command without a parameter. This list is quite long and scrolls by faster than you can read. Try piping the output of the command through DOS's More filter with the following:

```
C>mem /classify | more
```

```
Conventional Memory :

  Name          Size in Decimal      Size in Hex
  MSDOS           14160      ( 13.8K)      3750
  SETVER            400      (  0.4K)       190
  HIMEM            1184      (  1.2K)       4A0
  EMM386           8400      (  8.2K)      20D0
  COMMAND          2624      (  0.1K)       A40
  FREE               64      (  0.1K)        40
  FREE           628304      (613.6K)     99650

Total FREE  :    628368      (613.6K)

Upper Memory :

  Name          Size in Decimal      Size in Hex
  SYSTEM          180224      (176.0K)     2C000
  DOSKEY            4128      (  4.0K)      1020
  FREE               160      (  0.2K)        A0
  FREE             12032      ( 11.8K)      2F00

Total FREE :       12192      ( 11.9K)

Total bytes available to programs (Conventional+Upper) :       640560    (625.5K)
Largest executable program size :                              628128    (613.4K)
Largest available upper memory block :                          12032    ( 11.8K)

   1441792 bytes total EMS memory
   1048576 bytes free EMS memory

   3145728 bytes total contiguous extended memory
         0 bytes available contiguous memory
   1916928 bytes available XMS memory
         MS-DOS resident in High Memory Area
```

Figure 9-3. *The output of* mem /classify.

In the top part of the display are three columns: Name, Size in Decimal, and Size in Hex. Values are listed in decimal, by their "K" value, or in hexadecimal. (Refer to Chapter 1 or Appendix E for more information on hexadecimal.)

Two lists of programs are provided. The first includes programs and memory usage for conventional memory. The second list shows how the upper memory area is used.

The Name column gives the name of a program loaded into memory or generally describes how a hunk of memory is used. The first program listed in conventional memory is MSDOS, the MS-DOS system and boot program. This is followed by HIMEM and EMM386, two device drivers loaded into memory. Finally comes COMMAND, that last "program" loaded into memory. At the end of the first list are two lines labeled FREE, which is the amount of memory available.

In the second part of the Mem command's output, you can see programs that have been loaded into the upper memory area, specifically the *upper memory blocks*. When DOS is loaded high, it appears as SYSTEM in the list. Next comes the Doskey program. Finally, two areas (blocks) of free memory exist, both labeled FREE.

Under the Size columns you can see how much memory each program occupies. In Figure 9-3, HIMEM (actually HIMEM.SYS) uses up 1184 bytes of memory, or 1.2K, which is 4A0 hexadecimal. These size values are important. You'll see how they come into play later in this chapter, when the Devicehigh command is introduced.

Below the upper memory summary is a list of how all memory is used in the computer. There are 640,560 bytes (625.5K) available for DOS programs, the largest of which can be 628,128 bytes (613.4K). And the largest block available in upper memory is 12,032 bytes (11.8K).

That final value, the "largest available upper memory block," is used for comparison purposes. One strategy is to compare the size of that block with the size of some device driver or memory-resident program listed under "Conventional Memory." If the program is small enough, it can be transferred into that upper memory block. That's the secret of unlocking memory with DOS 5.

Getting DOS Out of Conventional Memory

DOS 5.0 is the first version of DOS to build memory-enhancing abilities into the operating system. The drawback is that these memory-enhancement tricks work only on 286 and later systems, and they work best on 386s and later. If you lack such a system or aren't currently running DOS version 5, skip ahead to the section "Doing It Without DOS 5."

If you have DOS 5, you can get more memory to run programs by moving DOS out of conventional memory and into the HMA, then moving device drivers and memory-resident software out of conventional memory and into the upper memory area, where special *upper memory blocks* are created. DOS 5 comes with all the tools to let you do this, provided you have a 286 or later microprocessor and at least 350K of extended memory.

If You Have a 286 or Later System

This section describes DOS 5's memory-management techniques for a PC with a 286 or later microprocessor and at least 350K of extended memory. If you have an earlier system, skip ahead to the section "Doing It Without DOS 5."

The 80286 and 80386 microprocessors can access megabytes of memory, many times more than an 8088 PC. That memory is extended memory, which has limited potential under DOS—unless you take advantage of memory device drivers.

There are two primary steps to getting access to the maximum amount of conventional DOS memory under DOS 5.0:

1. Load HIMEM.SYS to establish an XMS driver and create an extra 64K for use with DOS.

2. Transfer DOS into the HMA.

Additionally, you can use an expanded memory manager to convert extended memory into expanded memory. If you have a 386, you can use the memory manager supplied with DOS 5. The 286 cannot convert extended into expanded memory; you must buy expanded memory hardware, or make sure your system comes with some sort of LIM 4.0–based hardware, and then use special memory managers from third parties. Refer to the section "Doing It Without DOS 5" later in this chapter for more information.

Installing HIMEM.SYS

The first step in controlling extended memory under DOS is to install an extended memory system (XMS) memory device driver. The one that comes with DOS is HIMEM.SYS, which is installed into your CONFIG.SYS file using the following Device command:

```
device=c:\dos\himem.sys
```

Be sure to specify a path that gives the exact location of the HIMEM.SYS driver. Above, it's assumed HIMEM.SYS is in the \DOS subdirectory on drive C. Specify another location, such as *device=c:\himem.sys*, if the driver is in a different directory.

There are several optional parameters for HIMEM.SYS. They control specific aspects of the memory driver and should be used only if an application program or your PC's hardware manual suggests it. Otherwise, the simple command *device=c:\dos\himem.sys* in your CONFIG.SYS file will work the miracle.

HIMEM.SYS sets up the XMS standard on your PC, controlling all of extended memory and acting as a control program for other memory managers. It also allots the first 64K of extended memory as HMA, for use by DOS directly.

You should always install HIMEM.SYS if you're using any PC with extended memory, unless you're using a non-DOS 5 device driver that already supplies XMS support. Otherwise, provided you have an 80286 or later with at least 350K of extended memory, installing HIMEM.SYS is the first step to freeing up DOS's conventional memory.

Loading DOS high

HIMEM.SYS's biggest benefit to DOS users is an extra 64K of memory in which to play. (It also opens the door to extended memory for disk caches and RAM drives.) Most programs ignore the extra 64K of memory, which means you can use it to store DOS's own resident programs, all of which fit neatly into that 64K high memory area.

You can move DOS into the HMA with the Dos configuration command. To use the Dos configuration command, you must have the *device=himem.sys* command elsewhere in your CONFIG.SYS file.

The format of the Dos command, which was covered in the previous chapter, is as follows:

dos=*high/low,umb/noumb*

high/low tells DOS whether or not to load into the HMA. When you specify *dos=high*, DOS loads itself into the HMA. When you specify *dos=low*, DOS behaves as it normally does, loading itself into conventional memory. The default value for the command is *dos=low*, but Setup modifies your CONFIG.SYS file in such a way that the value of the command is *dos=high* if it detects extended memory.

umb/noumb controls upper memory blocks. These are areas of unused reserved memory that can be accessed only on a 386 or later system using the EMM386.EXE memory device driver. You don't need to specify a value for the *umb/noumb* option with a 286 computer, and if you do, it will be ignored (nothing personal). The default is *dos=noumb*.

To put DOS into the HMA, you include the line *dos=high* in your CONFIG.SYS file, in addition to the *device=c:\dos\himem.sys* command. This is all that's required to move DOS from conventional memory to the HMA.

Upper memory blocks, however, can be confusing. Although the upper memory area (reserved memory) normally contains ROM (BIOS), unused portions exist. Those portions can be accessed as RAM when you include the line *dos=umb* in your CONFIG.SYS file. The only reason to do this would be if you were going to use the Loadhigh command (covered later in this chapter). Also, you need at least 350K of extended memory to use the upper memory blocks.

To get the most out of memory on a 386 system, the best route to take is to combine options with the Dos configuration command as follows:

dos=high,umb

The above command will let you move most of DOS, device drivers, and memory-resident software out of your conventional memory.

Notice that since *low* and *noumb* are the defaults, it isn't necessary to put the line *dos=low,noumb* in a CONFIG.SYS file. (If a DOS professional finds it in your CONFIG.SYS file, you'll probably be shot.)

Edit your CONFIG.SYS file using a text editor or word processor, and include the following lines:

```
device=c:\dos\himem.sys
dos=high,umb
```

Remember to specify a full path for the HIMEM.SYS driver, such as *device=c:\himem.sys*, if it's not in your DOS directory. The line *dos=high, umb* will move DOS's memory-resident files from conventional memory to the HMA, giving you that much more memory in which to run applications.

Save your CONFIG.SYS file and return to DOS. Now enter the Mem command and take note of the value it displays as the *largest executable program size*. Reset your system with Ctrl-Alt-Del.

After booting, the changes made in your CONFIG.SYS file will take effect. You'll see something like the following message:

```
HIMEM: DOS XMS Driver Version x.xx - xx/xx/xx
XMS Specification Version 2.0
Copyright 1988-1991 Microsoft Corp.

Installed A20 handler number 1.
64K High Memory Area is available.
```

This indicates that HIMEM.SYS has been installed. Now run the Mem command again. There will be a marked increase in the amount of memory available.

If You Have a 386 or Later System

Among the many advanced features of 80386 and later microprocessors are memory-control facilities that enable them to do amazing things with

memory. You need no extra hardware to control the memory in your 386 or later PC; you just need the memory. DOS 5's memory-configuration commands directly tap the microprocessor's memory-control facilities.

With a 386 or later microprocessor, you can take two additional steps after installing HIMEM.SYS and specifying the Dos configuration command to get the maximum possible amount of conventional DOS memory:

1. Set up and install EMM386.EXE to convert extended into expanded memory and create upper memory blocks.
2. Transfer device drivers and memory-resident programs into expanded memory or upper memory blocks using the Devicehigh and Loadhigh commands.

These steps are not required. If you don't need the expanded memory driver, you could stop after loading HIMEM.SYS and moving DOS into the HMA. But for maximum memory savings, you'll want to install EMM386.EXE and use the Devicehigh or Loadhigh command.

Converting extended memory to expanded memory

A few applications can use extended memory on a 386 or later machine. For example, Windows 3.0 would like to see acres and acres of extended memory and only extended memory. But the majority of DOS applications (which don't run under Windows) can use only expanded memory if they need extra memory. Starting with version 5.0, DOS includes a device driver to convert extended memory to expanded memory on an 80386 or later microprocessor.

EMM386.EXE is a device driver, despite its EXE extension. It actually has two modes: It is both a device driver you install in CONFIG.SYS and a memory utility you use at the command prompt. As a device driver, EMM386.EXE is set up using the following configuration command:

device=c:\dos\emm386.exe *memory options noems ram*

memory is the amount of extended memory you want EMM386.EXE to convert into expanded memory. The value ranges from *16*, for 16K, through as much extended memory as you have in your system. If *memory* isn't specified, 256K is assumed.

options represents an armada of optional switches and parameters that customize how the MM386.EXE driver controls memory. You must specify *options* only in rare circumstances when you're experiencing problems, or when some application makes suggestions about what to specify.

noems is used when you want to use upper memory blocks but don't want any expanded memory. This is the switch you'd specify if you were running Windows under DOS 5.

ram tells EMM386.EXE to create upper memory blocks in addition to converting extended to expanded memory. You should not specify both *noems* and the *ram* parameters.

EMM386.EXE supplies the expanded memory system (EMS) manager. But in DOS 5, you must first load the HIMEM.SYS driver, under which EMS runs. To set up the extended memory in your PC, establish the HMA, move DOS there, and then allocate 256K of extended memory as expanded memory, put the following three lines in your CONFIG.SYS file:

```
device=c:\dos\himem.sys
dos=high
device=c:\dos\emm386.exe
```

That frees up quite a bit of RAM for your applications and sets up DOS to interact with both extended and expanded memory.

When used as a command-line program, EMM386.EXE tells you about the expanded memory in your system. Figure 9-4 shows the output of *emm386* when it is entered on the command line:

```
MICROSOFT Expanded Memory Manager 386  Version 4.20.06X
(C) Copyright Microsoft Corporation 1986, 1990

  Available expanded memory . . . . . .  256 KB

  LIM/EMS version . . . . . .  4.0
  Total expanded memory pages . . . . . .  40
  Available expanded memory pages . . . . .  16
  Total handles . . . . . .  64
  Active handles  . . . . . .  1
  Page frame segment  . . . . . .  D000 H

EMM386 Active.
```

Figure 9-4. *EMM386.EXE's command-line output.*

Granted, only a programmer would get enthusiastic over the information EMM386.EXE presents. *Memory pages* and *handles* in particular seem quite obscure. But when you access the upper memory area using the EMM386.EXE driver, additional information is displayed that tells you if and how you're using expanded memory. This information comes into play when you are tracking down errors.

Relocating device drivers

The final step in clearing out conventional memory is to take advantage of the upper memory blocks in the upper memory area. Normally, that memory is used only for ROM. But "holes" exist between the various ROMs and BIOS's. Using the memory-management abilities of the 80386, plus the XMS and EMS memory device drivers supplied with DOS, you can fill in those holes with RAM that can hold device drivers and memory-resident programs. With those items cleared out of conventional memory, you can, in some cases, have almost the full 640K free for applications.

The secret to this trick is the UMBs, upper memory blocks, that exist in the upper memory area. Figure 9-5 shows the holes in the upper memory area that can be remapped to contain usable RAM.

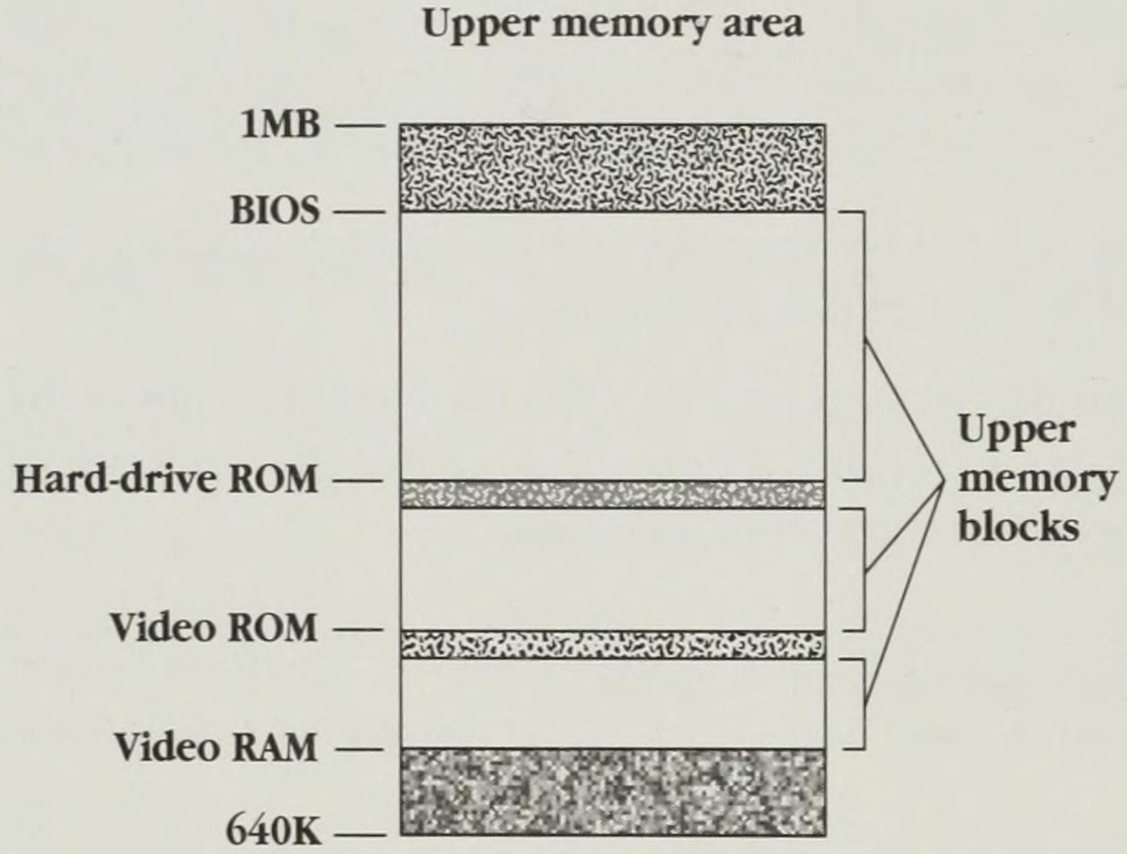


Figure 9-5. *Upper memory blocks are unused areas of memory in the upper memory area.*

DOS can remap extended memory into available spaces in the upper memory area. You do this by installing the HIMEM.SYS device driver, specifying the Dos configuration command with the *umb* option, and specifying the EMM386.EXE device driver with either its *ram* or its *noems* option. (The *ram* option converts some extended memory to expanded memory as well as creating the upper memory blocks; *noems* simply creates upper memory blocks.) You must have the following three lines in your CONFIG.SYS file:

```
device=c:\dos\himem.sys
dos=high,umb
device=c:\dos\emm386.exe ram
```

HIMEM.SYS must be installed before EMM386.EXE, but the Dos configuration command can be placed anywhere in CONFIG.SYS.

Using the Devicehigh command

After you set the above memory device drivers and settings, the next task is to move your device drivers into the upper memory blocks. You do this by substituting the Devicehigh command for Device in CONFIG.SYS. For example, suppose you're loading both the ANSI.SYS and MOUSE.SYS device drivers. Instead of using the Device command, use Devicehigh as follows:

```
devicehigh=c:\dos\ansi.sys
devicehigh=c:\mouse.sys
```

Those programs will now be loaded into available space in the upper memory area—into upper memory blocks. You can even move the EMM386.EXE device driver itself into high memory with the command *devicehigh=c:\dos\emm386.exe ram*. But note that some device drivers, such as SMARTDRV.SYS, should not be loaded high; refer to each driver's manual before you attempt to load it into upper memory.

Building a startup disk

To take advantage of the tricks mentioned thus far, you can edit your CONFIG.SYS file and reset your machine. But doing so may lead to incompatibilities and other problems. A better idea is to create a startup floppy disk on which you can test these techniques. To create such a disk, locate a new or available floppy and put it into your A drive. Format it using the following command:

```
C>format a: /s
```

The Format command formats the disk and transfers the DOS 5 system files to it. Next, you need to copy your memory device drivers to the disk, along with any other device drivers or memory-resident software you may want to experiment with. For example, you could enter:

```
C>copy c:\dos\himem.sys a:
     1 file(s) copied

C>copy c:\dos\emm386.exe a:
     1 file(s) copied

C>copy c:\dos\ansi.sys a:
     1 file(s) copied
```

```
C>copy c:\dos\doskey.com a:
     1 file(s) copied

C>copy c:\dos\mem.exe a:
     1 file(s) copied

C>_
```

In the end, five files are copied to drive A:

HIMEM.SYS
EMM386.EXE
ANSI.SYS
DOSKEY.COM
MEM.EXE

In addition, you might want to copy any other device drivers or memory-resident software you plan on loading into the upper memory area.

Keep the disk in drive A and press Ctrl-Alt-Del to reset. In a few moments, your system will boot from drive A. Since you haven't written an AUTOEXEC.BAT file, the system will ask you for the date and time and then display the DOS prompt:

```
A>
```

Enter the Mem command. Take note of the first three values, which may look something like this:

```
A>mem

    655360 bytes total conventional memory
    655360 bytes available to MS-DOS
    593296 largest executable program size
```

The final value you see above, as well as any extended memory reported, will be different for each system.

It's now time to start boosting conventional memory. Use Editor, your word processor, or *copy con* to create the following CONFIG.SYS file:

```
1: REM This is a test configuration file
2:
3: device=a:\himem.sys
4: dos=high,umb
```

Save the file on disk, then reset again with Ctrl-Alt-Delete, keeping the disk in drive A. This time, as the computer boots, you'll see information from the HIMEM.SYS driver displayed. Enter the date and time. Now try the Mem command again:

```
A>mem

    655360 bytes total conventional memory
    655360 bytes available to MS-DOS
    610320 largest executable program size
```

In the sample output above, there's an increase of 17,024 bytes in available conventional memory. Additionally, you'll see information on your screen about the XMS driver as well as the message *MS-DOS resident in High Memory Area.* If you didn't see such a message, if there was no memory improvement, or if you saw an error, go back and reedit your CONFIG.SYS file. Reset again and you should see the improvement.

To take advantage of the Devicehigh command, add the following two lines to your CONFIG.SYS file on drive A:

```
5: device=a:\emm386.exe ram
6: devicehigh=a:\ansi.sys
```

Save your CONFIG.SYS file, then reset again by pressing Ctrl-Alt-Del. You'll see additional messages as the computer boots, this time including one from the EMM386.EXE file as it converts 256K of extended memory to expanded memory (the default) and activates the upper memory blocks with its *ram* switch. If you see an error message when using the Devicehigh command, you probably forgot either the *umb* option with the Dos configuration command or EMM386.EXE's *ram* switch.

Enter the date and time, then use the Mem command again and look at the results. Compare them with the previous values. You may notice a lower value because EMM386.EXE has been installed into conventional memory. But is ANSI.SYS really in an upper memory block?

To prove it, use the command *mem /classify* and scan for the device driver's name. You'll see it located in the Upper Memory part of the Mem command's output.

If you're experimenting with your own device drivers, you can install them now as well. You might want to start with the above bare minimum, then add your own device drivers one at a time. This approach will aid in troubleshooting if anything weird happens.

When you're satisfied everything works, incorporate the additions into your main CONFIG.SYS file on your hard drive. (Do not copy the CONFIG.SYS file from drive A to drive C!) Remember to change full path names where applicable. Then reset to put the changes into effect.

Beware of expanding device drivers

A problem may occur with some device drivers that *expand* after they've been loaded into memory. Though DOS does handle the relocation of most device drivers, some will grow like rising dough in memory after they're installed, others may expect to be located below 640K, and still others just may not work. To control these swelling device drivers, you should specify the *size* parameter when using the Devicehigh command.

Specifying a size for a program is tricky. It involves two steps:

First, load the program as you normally would, using the Device command. Then, at the DOS prompt, use the Mem command to locate the file in memory. Take note of its size. For example, suppose the RAMPIG.SYS device driver has been loaded. The program appears in the output of *mem /classify | more* as follows:

```
RAMPIG        12736    (12.4K)     31C0
```

The values indicate RAMPIG's actual size in memory. Because the Devicehigh command understands only hexadecimal, you need to take note of the final value, 31C0.

The second step is to edit CONFIG.SYS to reflect the true size of the device driver. You will change:

```
device=c:\rampig.sys
```

to the following:

```
devicehigh size=31c0 c:\rampig.sys
```

❑ **NOTE:** *A worse problem may occur with device drivers that initially expand but then shrink after they're loaded. The initial expansion takes place before COMMAND.COM is loaded, so the Mem command won't reflect the ballooning. This type of device driver is rare, but there might be one lurking out there somewhere. Refer to your device driver's manual to find out if this expansion/reduction takes place.*

The *size=31c0* part of the command tells CONFIG.SYS just how much RAMPIG.SYS will grow. This aids DOS when it relocates the device driver into an upper memory block. But keep in mind that you need to do this for only a few device drivers. Otherwise, the *size* option isn't needed.

Relocating memory-resident software

To load memory-resident software (TSRs) from AUTOEXEC.BAT into the upper memory area, the Loadhigh, or LH, command is used. This command has the same effect as Devicehigh, but it can be used at the DOS prompt or in a batch file to move memory-resident software into upper memory blocks.

The format of the Loadhigh command is:

loadhigh *filename options*

filename is the name of a program that you want to load into the upper memory area. It must be a memory-resident program, a startup utility, or some other program that stays in memory. Examples of DOS programs that work with Loadhigh are Doskey, DOSShell, Keyb, and Print. Non-DOS programs include SideKick and other memory-resident or "pop-up" utilities. Specifying a non-memory-resident program will have unpredictable results.

options are the options and parameters normally specified after whatever program is named in *filename*. In fact, a good way to look at the Loadhigh command is to remember that you simply stick it in front of the normal command to load a memory-resident utility you want to load high.

Loadhigh also has an abbreviated form, *LH*, which can be used instead of Loadhigh just as you can substitute CD for Chdir in the Change Directory command.

Like the Devicehigh configuration command, Loadhigh requires HIMEM.SYS to be loaded, the DOS command to be set with the *umb* option, and EMM386.EXE to be loaded with either the *ram* or the *noems* option. Specifying *umb* and *ram* or *noems* is most important; Loadhigh won't work without them.

To put Loadhigh to work, edit your AUTOEXEC.BAT file and place the Loadhigh command in front of each memory-resident program's name. For example, if you're using the Doskey command-line enhancer, edit the line to read:

```
loadhigh c:\dos\doskey
```

That's it! Now Doskey is loaded into the upper memory area, and you have more memory available to run your applications. To prove it, run the Mem command with the */classify* option and note both the relocated program and the increased memory available.

As with the Devicehigh command, you might want to create an AUTOEXEC.BAT file on a separate boot disk. Experiment with Loadhigh and your memory-resident software until you're certain everything works correctly. Then incorporate those changes back into the original AUTOEXEC.BAT file on your hard drive.

One last thing to remember: Loadhigh and Devicehigh move only the thinking part of a memory-resident program or device driver. For example, if you load a RAM disk using Devicehigh, only the driver will be placed in an upper memory block; the RAM drive itself will still be in extended or expanded memory. So don't hesitate to load a device driver or memory-resident program into an upper memory block because it establishes a 2K print spooler or a 512K disk cache; those items will be placed elsewhere in memory.

Doing It Without DOS 5

The DOS 5 solution is intended for 386 and later PCs, which makes sense in the nineties: The 8088 is long gone from the business and professional scene (you can hardly find the machines anymore), and the 80286 is rapidly being replaced by the economical 386SX chip. If you're stuck with an older PC, however, you don't need to give up hope.

DOS 5 does support the 80286 chip, but it can't convert extended memory to expanded memory or relocate device drivers and memory-resident programs into the upper memory area on 286 computers. If you have an 8088 chip, you don't have extended memory at all; therefore, you don't need HIMEM.SYS and cannot use any of its capabilities. Solutions exist, however, that give 8088 and 80286 PCs power similar to that described earlier in this chapter for the 80386 chip. The strategy is basically the same, but the software that does the job works differently.

The object of the game is to give your machine as much *expanded* memory as possible. For an 8088, you should buy an expanded memory board and pack it full of memory. For a 286, even if you already have extended memory, you must specifically add expanded memory; there is no conversion process.

On both machines, it helps to disable as much of the computer's main, or motherboard, memory as possible. Then, using the memory expansion hardware, *backfill* memory from the expansion card into main memory. This technique will depend on the memory hardware you use. The strategy is to disable all but about 256K of main memory and then use the memory expansion hardware to supply main memory from 256K through 640K and beyond.

Once you've installed the memory, you need a memory device driver. That driver sets up the EMS standard on an 8088-based system and sets up both the EMS and XMS standards on an 80286-based system.

Currently, two popular third-party packages exist to do the job: QRAM (pronounced *cram*) from Quarterdeck Office Systems and MOVE'EM from Qualitas. Both programs will control extended and expanded memory, supply XMS and EMS drivers, and give you substitutes for the Devicehigh and Loadhigh commands. Previously in this chapter, the following configuration commands were used to set up a 386 PC to get the most from memory under DOS 5:

```
device=c:\dos\himem.sys
dos=high,umb
device=c:\dos\emm386.exe ram
devicehigh=c:\dos\ansi.sys
devicehigh=c:\mouse.sys
```

On an 80286 system running QRAM, the following commands (in italics) are substituted:

```
device=c:\himem.sys
device=c:\emm.sys at 258
device=c:\qram\qram.sys r:1 ram
dos=high
device=c:\qram\loadhi.sys /r:1 c:\dos\ansi.sys
device=c:\qram\loadhi.sys /r:1 c:\mouse.sys
```

HIMEM.SYS is loaded first. An expanded memory manager, EMM.SYS, is installed next, setting up the expanded memory board in the computer. Then QRAM is installed, supplying the EMS device driver.

Note that the Dos configuration command is still used. However, there's no need to specify the *umb* option; that's handled by QRAM. Similar commands in MOVE'EM do essentially the same things.

These programs are also available for the 80386 system. The following is a CONFIG.SYS file using 386MAX from Qualitas. Under DOS 5, this configuration leaves 639K of conventional memory free according to the Mem command:

```
device=c:\386max\386max.sys pro=c:\386max\386max.pro
device=c:\386max\386load.sys prog=c:\mouse\mouse.sys /c1
device=c:\386max\386load.sys prog=c:\dos\ansi.sys
```

Note that not all DOS 5's memory configuration commands will work in these third-party configurations. Although you can use QRAM or 386MAX, the Devicehigh and Loadhigh commands might not be compatible with them. You can check the manuals to be sure, or just use the substitute commands that come with the third-party software.

These third-party memory drivers offer additional advantages over the memory-saving tools in DOS 5.0. (If they didn't, no one with a 386 would need to buy the programs; after all, these tricks are now free with DOS.) One of the more interesting options allows you to convert video memory on an EGA or a VGA system for use as conventional DOS memory. By pulling the right switches, you can have up to 736K free for running DOS programs.

There are limitations to these techniques, including those offered with DOS 5. Not all applications are happy being relocated into an upper memory block. And some applications are fussy, wanting expanded memory when you have extended memory, or vice versa.

CHAPTER

10

Setting Up and Using a RAM Disk

A RAM disk is an area of memory that behaves like another disk drive attached to your computer. You don't need any additional hardware to install a RAM disk, just a program that persuades DOS to treat a portion of memory as a disk drive—a disk drive that happens to be very fast.

Starting with version 3.1, DOS includes such a program, called RAMDRIVE.SYS. (The IBM release of DOS includes a similar program, called VDISK.SYS.) Most add-on memory cards also include a RAM disk program. Even if you don't have a RAM disk program, you still might want to skim this chapter just to see what you're missing.

It's called a RAM disk because the computer's memory is commonly called RAM, for *random access memory.* This term is a tenacious misnomer. Access to memory, in fact, is quite the opposite of random; if it weren't, finding a program or data in memory would be practically impossible. But such misnomers have precedents in everyday language. A city named Mount Pleasant, after all, may not be on a mountain nor be remarkably pleasant. So *RAM disk* it is—a short, punchy name that's easier to say (and remember) than *electronically simulated rotating memory device.*

You can use a RAM disk just as you would use a real disk drive because, as far as DOS is concerned, a RAM disk is just another disk drive. You can create files and subdirectories on it, copy files to and from it, even run the Check Disk (chkdsk) command on it. And a RAM disk is *really* fast, much faster than a real disk drive, because it has no moving parts. Using a RAM disk not only saves time, it also saves wear and tear on your real disk drives.

So what's the catch? Well, a RAM disk requires memory. Because the memory used by a RAM disk isn't available to DOS or other programs, your system must have enough memory to run DOS and all your application programs without using the memory you assign to the RAM disk.

And mark this well: The contents of most RAM disks are lost when power is removed from the system or when you restart DOS. So if there is a power failure, you lose any work you have done using the RAM disk that you didn't copy to a real disk. More to the point, it's all too easy to work for several hours and then, relieved at finally finishing, shut down without copying the changed files to a real disk. It's difficult to describe the emotions that follow when you realize what you have done. Try to avoid learning this firsthand.

Defining a RAM Disk

You can define a RAM disk if your system can spare the memory and your version of DOS includes either RAMDRIVE.SYS or VDISK.SYS. The remainder of this chapter describes the use of RAMDRIVE.SYS. VDISK.SYS works in a similar manner. Defining a RAM disk involves putting a Device command that names the RAM disk program in CONFIG.SYS, then restarting the system. The command parameters specify the capacity, sector size, and maximum number of directory entries of the RAM disk; if your computer has expanded or extended memory, you can also tell DOS to put the RAM disk in that added memory, saving conventional memory for DOS and your application programs.

The RAM disk program assigns the next available sequential disk drive letter to the RAM disk. For example, if you have two floppy-disk drives (named A and B), the RAM disk would be drive C; if you have a hard drive (C), the RAM disk would be drive D. Other RAM disk programs may require you to assign a drive letter to your RAM disk.

If you have a different RAM disk program than RAMDRIVE.SYS, such as SUPERDRV.COM from AST Research, Inc., the details of the command description here won't necessarily apply to your program. Check the manual that came with your RAM disk program for differences.

Using RAMDRIVE.SYS

The DEVICE=RAMDRIVE.SYS command has five parameters:

device=ramdrive.sys *size sector directory* /e /a

size is the capacity of the RAM disk in kilobytes. The minimum is 16K, and the maximum is 4096K, or 4MB. Be sure to leave enough memory to run DOS and your application programs. If you don't specify a value for *size*, a value of 64 is assumed.

sector is the size of each sector in bytes. You can specify 128, 256, or 512. A sector on a real disk, regardless of type, contains 512 bytes. If you specify an incorrect value, DOS sets *sector* to 512.

directory specifies the number of root-directory entries allowed on the RAM disk. You can specify any value from 2 through 1024. Each directory entry takes up 32 bytes of the RAM disk; DOS increases the number of directory entries, if necessary, to make the directory occupy an integral number of sectors. If you omit the value or specify an incorrect value, DOS sets *directory* to 64.

/e tells DOS to use extended memory above 1MB for the RAM disk. This parameter is valid only if you're using a computer with extended memory installed. A RAM disk in extended memory doesn't reduce the amount of memory available to DOS or application programs.

/a tells DOS to use expanded memory for the RAM disk. This parameter is valid only if your computer contains an expanded memory board or uses an expanded memory emulator that meets the Lotus-Intel-Microsoft (LIM) expanded memory specification. A RAM disk in expanded memory doesn't reduce the amount of memory available to DOS or application programs.

If you're using either the /e or the /a switch make sure you have installed an XMS or EMS memory device driver in CONFIG.SYS *before* you set up the RAM disk. For /e, make sure HIMEM.SYS or another XMS driver has been installed; for /a, make sure EMM386.EXE or a similar EMS driver has been installed before you use the *device=ramdrive.sys* command.

Examples of defining a RAM disk

Assume that the RAM disk program is in the \DOS directory on your hard drive (drive C). To define a 128K RAM disk with 256-byte sectors and room for 64 root-directory entries, you would include one of the following commands in CONFIG.SYS:

```
device=c:\dos\ramdrive.sys 128 256 64
```

To define a 1024K RAM disk with 512-byte sectors and room for 200 directory entries in extended memory, you would include one of the following commands in CONFIG.SYS:

```
device=c:\dos\ramdrive.sys 1024 512 200 /e
```

You can define multiple RAM disks by including multiple DEVICE=RAMDRIVE.SYS commands in CONFIG.SYS. Each additional RAM disk will be assigned the next available drive letter.

Using a RAM Disk

To take advantage of the speed of the RAM disk, use the Copy command to copy the data files, batch files, and programs you'll be working with to the RAM disk. Be sure to leave enough unused space on the RAM disk, however, to allow for the increased size of files you edit and any backup files or temporary files that your application programs may create.

To simplify setting up the RAM disk, write one or more batch files to copy the files and programs you need to work on a specific project. If you consistently use the same setup, put the Copy commands in AUTOEXEC.BAT so that the RAM disk will be ready to use when DOS first displays the system prompt.

> ❑ **NOTE:** *You may not be able to use a copy-protected program from a RAM disk; check the program's manual for instructions or warnings.*

If you use a RAM disk regularly, a couple of cautions are in order:

- Back up your work frequently to a real disk. A power failure—or forgetting you're using a RAM disk and turning off the system—means that all your work is lost.
- Leave room for enough directory entries. Using all the directory entries has the same effect as filling the disk; you won't be able to add any new files without getting a *Disk full* error message—even if you have the space. Remember, most word processors and text editors create backup files, so you'll need up to twice as many directory entries as the number of files you work on. Some application programs create temporary files that they delete before returning to DOS; each temporary file also requires a directory entry. A directory entry requires only 32 bytes, so you can safely specify 64 directory entries; that would take up only 2K of the RAM disk and should avoid the problem of filling the disk. If you'll be working with many small files or using several different application programs, specify at least 128 directory entries.

Creating a RAM Disk with a Batch File

There may be times when you'd like to define another RAM disk, or maybe work without the RAM disk for a while so that another program could use the memory, but it's too much trouble to change CONFIG.SYS and restart DOS. You can avoid most of this trouble with the batch file described here—RAMDRIVE.BAT—which automates creating a RAM disk: You tell RAMDRIVE.BAT what sort of RAM disk you want, and RAMDRIVE.BAT takes care of changing CONFIG.SYS and restarting DOS.

You can use RAMDRIVE.BAT to add a RAM disk, delete all RAM disks, or delete all currently defined RAM disks and start over with a different RAM disk. Before RAMDRIVE.BAT restarts DOS to carry out your instructions, it gives you a chance to change your mind. If you enter *RAMDRIVE* with no parameters, it displays instructions and tells you what RAM disks are currently defined.

Restarting DOS with a Program

RAMDRIVE.BAT resets the computer with the program RESET.COM, which has the same effect as pressing Ctrl-Alt-Del. You don't have to use RESET.COM, but it's a small touch that makes RAMDRIVE.BAT more convenient. If you prefer not to use RESET.COM, skip this section and go to "Entering RAMDRIVE.BAT," which includes instructions for adding a prompt that tells the user when to press Ctrl-Alt-Del.

You create RESET.COM with Debug, using the procedure described in Chapter 5. If you haven't yet gone through the examples in Chapter 5, you might do so before continuing here.

Using your text editor, word processor, or Editor or Edlin, create a file named RESET.SCR and enter the lines shown in Figure 10-1.

```
 1:  a 100
 2:  mov ax,40
 3:  mov ds,ax
 4:  mov ax,1234
 5:  mov [0072],ax
 6:  jmp f000:e05b
 7:
 8:  r cx
 9:  10
10:  n reset.com
11:  w
12:  q
```

Figure 10-1. *RESET.SCR: Restarting DOS with a program.*

Proofread RESET.SCR against Figure 10-1 carefully, making certain that line 7 is blank and that the last line is *q*; if there is a mistake in either of these lines, you'll have to manually restart DOS when you try to create RESET.COM. When you're sure that RESET.SCR is correct, save it and enter the following Debug command:

```
C>debug < reset.scr
```

DOS should respond as shown in Figure 10-2.

If nothing seems to happen and the keyboard refuses to respond, restart DOS. Load RESET.SCR with your text editor or word processor and proofread it carefully against Figure 10-1. Correct any errors and reenter the Debug command.

```
-a 100
xxxx:0100 mov ax,40
xxxx:0103 mov ds,ax
xxxx:0105 mov ax,1234
xxxx:0108 mov [0072],ax
xxxx:010B jmp f000:e05b
xxxx:0110
-r cx
CX 0000
:10
-n reset.com
-w
Writing 0010 bytes
-q
```

Figure 10-2. *DOS response to creating RESET.SCR.*

If DOS displays a series of commands and responses after you enter the Debug command, proofread the display against Figure 10-2. If they don't agree, go back and edit RESET.SCR again; proofread it against Figure 10-1, correct any differences, and then save it, and reenter the Debug command.

Before you test RESET.COM, make sure your system is ready to shut down. In particular, if you're using a RAM disk, copy any files you have changed to a real disk. Now enter *reset*. The system should respond just as if you pressed Ctrl-Alt-Del.

Entering RAMDRIVE.BAT

RAMDRIVE.BAT is a long batch file; many of its 61 lines are devoted to checking for errors, displaying messages, and creating temporary files that allow you to change your mind at the last minute. Even if you don't think you want to use RAMDRIVE.BAT, its commands and the explanations that follow might give you some ideas to use in other batch files.

Figure 10-3 shows the commands in RAMDRIVE.BAT. If you have DOS 5 you can substitute *echo.* for *echo* <*Alt-255*> to echo a blank line.

```
1:    @echo off
2:    c:
3:    cd \
4:    if "%1"=="none" goto NONE
5:    if "%1"=="new"  goto NEW_OR_ADD
```

Figure 10-3. *RAMDRIVE.BAT: Automatically defining a RAM disk.* *(continued)*

Figure 10-3. *continued*

```
 6:    if "%1"=="add"  goto NEW_OR_ADD
 7:    echo <Alt-255>
 8:    echo [1mCommand                          Result[m
 9:    echo ramdrive add {size} {sec} {dir}  Add this RAM disk and restart
10:    echo ramdrive new {size} {sec} {dir}  Define just this RAM disk and restart
11:    echo ramdrive none                    Restart with no RAM disk
12:    goto STATUS
13: :NONE
14:    copy config.$$$ config.new > nul
15:    goto RESET
16: :NEW_OR_ADD
17:    if not "%2"=="" if not "%3"=="" if not "%4"=="" goto OK
18:    echo [7m You must specify at least 3 RAMDRIVE parameters. [m
19:    goto END
20: :OK
21:    if %1==new copy config.$$$ /a config.new /b > nul
22:    if %1==add copy config.sys /a config.new /b > nul
23:    if %1==add if exist ramdrive.log copy ramdrive.log log.new > nul
24:    echo device=c:\dos\ramdrive.sys %2 %3 %4 %5 >> config.new
25:    if "%5"=="" echo %2K disk with %3-
       byte sectors and %4 directory entries >> log.new
26:    if "%5"=="/e" echo %2K disk with %3-
       byte sectors and %4 directory entries in extended memory >> log.new
27:    if "%5"=="/a" echo %2K disk with %3-
       byte sectors and %4 directory entries in expanded memory >> log.new
28 :RESET
29:    cls
30:    echo              <Alt-7>[7m Warning [m
31:    echo If you continue, your system will be reset
32:    echo with the following RAM disks defined:
33:    echo <Alt-255>
34:    if exist log.new type log.new
35:    if not exist log.new echo ** None **
36:    echo <Alt-255>
37:    echo Press F1 to continue; any other
38:    echo response cancels this RAMDRIVE command.
39:    reply
40:    if errorlevel 59 if not errorlevel 60 goto DO_IT
41:    echo <Alt-255>
42:    echo ** Canceled **
43:    if exist log.new erase log.new
44:    erase config.new
45:    goto STATUS
46: :DO_IT
47:    attrib -r config.sys
48:    erase config.sys
49:    rename config.new config.sys
```

(continued)

Figure 10-3. *continued*

```
50:    attrib +r config.sys
51:    if exist ramdrive.log attrib -r ramdrive.log
52:    if exist ramdrive.log erase ramdrive.log
53:    if exist log.new rename log.new ramdrive.log
54:    if exist ramdrive.log attrib +r ramdrive.log
55:    reset
56: :STATUS
57:    echo <Alt-255>
58:    echo ▯[7m Active RAM Disks ▯[m
59:    if exist ramdrive.log type ramdrive.log
60:    if not exist ramdrive.log echo ** None **
61: :END
```

How RAMDRIVE.BAT works

RAMDRIVE.BAT uses two files to keep track of your RAM disks:

- CONFIG.$$$ contains all the commands you want in CONFIG.SYS except those that create a RAM disk (such as *device=ramdrive.sys*). RAMDRIVE.BAT copies CONFIG.$$$ to CONFIG.SYS when you delete all RAM disks. CONFIG.$$$ should have at least the line *device=ansi.sys* in it.
- RAMDRIVE.LOG contains a description of each RAM disk currently defined in CONFIG.SYS. The presence or absence of RAMDRIVE.LOG tells RAMDRIVE.BAT whether CONFIG.SYS contains a command that defines a RAM disk. If RAMDRIVE.LOG doesn't exist, RAMDRIVE.BAT assumes that no RAM disk is currently defined; if RAMDRIVE.LOG does exist, RAMDRIVE.BAT assumes that at least one RAM disk is defined.

Both the CONFIG.$$$ and RAMDRIVE.LOG files are in the root directory. You create CONFIG.$$$ by copying CONFIG.SYS before you use RAMDRIVE.BAT for the first time. RAMDRIVE.BAT creates and deletes the RAMDRIVE.LOG file as required.

Here's how RAMDRIVE.BAT works:

- Lines 2 and 3 change the current drive to drive C and the current directory to the root. If you aren't using a hard drive, change line 2 to *a:* (or to the drive from which DOS normally starts, if different).
- Lines 4–6 check the first parameter; if it is *none*, RAMDRIVE.BAT goes to delete all RAM disk definitions and restart DOS. If the parameter is *new* or *add*, RAMDRIVE.BAT goes to define a RAM disk. If it is none of those, RAMDRIVE.BAT continues with line 7.

- Lines 7–12 display instructions for using RAMDRIVE.BAT, then go to display the RAM disks currently defined and return to DOS. There are 26 spaces in line 8; line up the words *Add*, *Define*, and *Restart* in lines 9–11.
- Lines 14 and 15 delete all commands that define a RAM disk from CONFIG.SYS by copying CONFIG.$$$ to CONFIG.NEW, then go to reset the system. This restarts DOS with no RAM disks defined.
- Line 17 goes to define a RAM disk only if three parameters are specified after the first. This test is made because RAMDRIVE.BAT carries out this command only if the first parameter is *new* or *add* (checked in lines 5 and 6), and it takes at least three parameters to define a RAM disk.
- Lines 18 and 19 display an error message and return to DOS if you didn't specify three parameters after *new* or *add*.
- Line 21 makes a temporary copy of CONFIG.$$$ called CONFIG.NEW that contains no commands to define a RAM disk if the first parameter is *new*. This deletes any RAM disks currently defined.
- Line 22 makes a temporary copy of CONFIG.SYS called CONFIG.NEW that contains current RAM disk definitions if the first parameter is *add*. This adds the RAM disk being created to RAM disks currently defined.
- Line 23 makes a temporary copy of RAMDRIVE.LOG (if it exists) called LOG.NEW if the first parameter is *add*. Thus, the RAM disk being created will be added to any existing RAM disk, but will be the only RAM disk if none is currently defined.
- Line 24 adds a RAMDRIVE.SYS command that defines the RAM disk being created to the temporary copy of CONFIG.SYS created in line 21 or 22. As you'll see, the temporary file is used so that you can change your mind at the last minute and leave your current CONFIG.SYS file unchanged.
- Lines 25–27 write a description of the RAM disk being created in RAMDRIVE.LOG. Line 26 adds the words "in extended memory" if you include the /e parameter, and line 27 adds the words "in expanded memory" if you include the /a parameter (*%5* in the RAMDRIVE.BAT command). There's no other check for a specific value of *%5*, so if you enter anything other than */e* or */x* for this parameter, RAMDRIVE.BAT doesn't put the description in RAMDRIVE.LOG but does put the *device=ramdrive.sys* command in CONFIG.SYS. This causes a discrepancy between RAMDRIVE.LOG and CONFIG.SYS; you can correct it by entering another RAMDRIVE.BAT command that specifies either *none* or *new*.

❑ **NOTE:** *If you're going to use VDISK.SYS instead of RAMDRIVE.SYS, change the /a in line 27 to /x:*

```
if "%5"=="/x" echo %2K disk with %3-byte sectors and
   %4 directory entries in expanded memory >> log.new
```

- Lines 29–38 clear the screen, warn you that the system is going to be reset, show you the RAM disks that will be defined if you continue, and prompt you to press F1 to reset the system. Lines 34 and 35 show you the RAM disks that will be defined either by displaying RAMDRIVE-.NEW (the temporary version of RAMDRIVE.LOG) or—if there is no RAMDRIVE.NEW—by displaying ** *None* **.
- Lines 39 and 40 wait for you to press a key and go to reset the system if you press F1. (These lines require REPLY.COM in your command path or the current directory, which you created in Chapter 7.)
- Lines 41–45 display a blank line and ** *Canceled* **, erase any temporary files, then go to display the RAM disks currently defined, and return to DOS. Because all the changes so far have been limited to temporary files, CONFIG.SYS and RAMDRIVE.LOG are unaffected.
- Line 47 removes the read-only attribute from CONFIG.SYS. For RAMDRIVE.BAT to work properly, CONFIG.SYS and CONFIG.$$$ must be identical except for any commands that define RAM disks, and RAMDRIVE.LOG must contain an accurate description of the RAM disks defined in CONFIG.SYS. To ensure that the files remain accurate, they are kept read-only except when RAMDRIVE.BAT itself changes them.
- Lines 48–50 make the temporary copy of CONFIG.SYS the real thing by erasing CONFIG.SYS and by changing the name of CONFIG.NEW to CONFIG.SYS and make the new copy read-only so that it can't be changed inadvertently.
- Line 51 removes the read-only attribute from RAMDRIVE.LOG if it exists. This line and the next three check to see whether RAMDRIVE-.LOG exists, because there is no RAMDRIVE.LOG if a RAM disk isn't currently defined and DOS displays an error message if you type an Attribute command for a file that doesn't exist.
- Line 52 erases RAMDRIVE.LOG if it exists.
- Line 53 makes the temporary copy of RAMDRIVE.LOG (if there is one) the real thing by changing the name of LOG.NEW to RAMDRIVE.LOG.
- Line 54 makes the new copy of RAMDRIVE.LOG (if it exists) read-only.

- Line 55 resets the system. If you aren't using RESET.COM, replace reset with the following two commands:

```
echo Press <Ctrl-Alt-Del> to reset the system
reply
```

  If you make this change but don't reset the system at this point, RAMDRIVE.LOG won't accurately describe the RAM disks defined in CONFIG.SYS and RAMDRIVE.BAT won't work properly. Using RESET.COM guarantees that if you reply to the warning prompt with F1, the system will be reset and RAMDRIVE.LOG will be accurate.

- Lines 57–60 display the RAM disks currently defined by displaying RAMDRIVE.LOG or—if there is no RAMDRIVE.LOG—** *None* **.

Testing RAMDRIVE.BAT

Before you test RAMDRIVE.BAT, do the following to prepare the files it uses:

1. Make a backup copy of CONFIG.SYS named CONFIG.OLD.
2. If CONFIG.SYS contains any commands that define a RAM disk, delete them.
3. Copy CONFIG.SYS to CONFIG.$$$.
4. Enter *attrib +r config.** to make CONFIG.SYS, CONFIG.$$$, and CONFIG.OLD read-only.
5. If there is a file in the root directory named RAMDRIVE.LOG, rename or erase it.

RAMDRIVE.BAT assumes the first time you run it that CONFIG.SYS does not contain a RAM disk command (which should be correct) because there is no file named RAMDRIVE.LOG in the root directory.

If you created RAMDRIVE.BAT on a RAM disk, copy it to a real disk before proceeding. Now you're ready to test RAMDRIVE.BAT. Enter the command with no parameters:

```
C>ramdrive
```

RAMDRIVE.BAT should respond by displaying its instructions and any RAM disks currently defined:

```
Command                                 Result
ramdrive add {size} {sec} {dir}  Add this RAM disk and restart
ramdrive new {size} {sec} {dir}  Define just this RAM disk and restart
ramdrive none                           Restart with no RAM disk
```

```
 Active RAM Disks
** None **
```

Type the following to define a 64K RAM disk with 128-byte sectors and room for 32 directory entries:

```
C>ramdrive add 64 128 32
```

RAMDRIVE.BAT clears the screen, beeps, and warns you that if you continue it will reset the system:

```
                Warning
If you continue, your system will be reset
with the following RAM disks defined:

64K disk with 128-byte sectors and 32 directory entries

Press F1 to continue; any other
response cancels this RAMDRIVE command.
```

Press the Spacebar to cancel the command. RAMDRIVE.BAT displays:

```
** Canceled **

 Active RAM Disks
** None **

C>_
```

This time you'll go through with it. Enter the following to define a 128K RAM disk with 512-byte sectors and room for 64 directory entries:

```
C>ramdrive add 128 512 64
```

Again, RAMDRIVE.BAT warns you that if you continue it will reset the system. This time press F1. The system should restart and, in a few moments, display the system prompt.

To verify that things went as they should have, first enter *ramdrive*; RAMDRIVE.BAT should display its instructions, then a description of the RAM disk currently active:

```
128K disk with 512-byte sectors and 64 directory entries
```

All this really tells you is that the file RAMDRIVE.LOG contains this line. Verify the existence of the RAM disk by displaying its directory. If you

have only two floppy-disk drives, enter *dir c:*; if you're using a hard drive, enter *dir d:*, or whichever drive letter is the next highest after your last hard drive. You should see the empty root directory of the RAM disk.

Sometimes you might want more than one RAM disk. Type the following to add a 64K RAM disk with 256-byte sectors and room for 32 directory entries:

```
C>ramdrive add 64 256 32
```

After the warning, press F1. After the system resets and DOS displays the system prompt, type *ramdrive*. Now you should see two descriptions:

```
128K disk with 512-byte sectors and 64 directory entries
64K disk with 256-byte sectors and 32 directory entries
```

Verify the existence of this new RAM disk by displaying its directory. If you have only two floppy-disk drives, enter *dir d:*; if you're using a hard drive, enter *dir e:* (or whatever's next). You should see the empty root directory of the RAM disk.

Now decide what kind of RAM disk you would like to routinely use on your system, and create it with the *new* option of RAMDRIVE. For example, to create a 360K RAM disk with 512-byte sectors and room for 128 directory entries, you would enter *ramdrive new 360 512 128*. To create a 1MB RAM disk in extended memory with 512-byte sectors and room for 200 directory entries, you would enter *ramdrive new 1024 512 200 /e*. To operate with no RAM disks, you would enter *ramdrive none*.

Some closing points about RAMDRIVE.BAT

Although RAMDRIVE.BAT checks to make sure that you enter at least three parameters, it has no way of knowing that you have entered correct parameters. It accepts *ramdrive add mairzie doats and doazie doats* as readily as *ramdrive add 128 256 32 /e*. DOS might not be able to execute such a command, so be sure to double-check the final verification that RAMDRIVE.BAT displays before you press F1 to reset the system.

Unless you tell it otherwise, DOS recognizes only drive letters *a* through *e*. If you think you might need more drives than that, put a Lastdrive configuration command in CONFIG.SYS; *lastdrive=g*, for example, would let you use four RAM disks on a system with one floppy drive and a hard drive, or five RAM disks on a two-floppy-drive system.

Remember that the root directory must contain the file CONFIG.$$$, a duplicate of CONFIG.SYS with no RAM disk commands. So, if you change CONFIG.SYS after you start using RAMDRIVE.BAT, be sure that you copy

CONFIG.SYS to CONFIG.$$$ after you make the changes; if you don't do this, the changes you made to CONFIG.SYS will be lost the first time you use RAMDRIVE.BAT with either the *new* or the *none* option.

And it's worth repeating two cautions:

- Be sure to back up your work frequently from a RAM disk to a real disk to protect yourself from your own forgetfulness as well as from power failures.
- If you're going to be working with many files or several application programs, specify at least 128 directory entries on your RAM disk.

You wouldn't want to lavish the sort of effort represented by RAMDRIVE.BAT on all your batch files, but it shows just how much you can do with DOS. Taking the extra time to anticipate mistakes, display informative or reassuring messages, and provide graceful escape routes adds touches that even some professional application programs lack.

And after using a batch file that looks good while it saves you time, it feels good to lean back and say, "I did that."

Speeding Up Disk Performance with a Cache

A RAM disk is an interesting solution. It provides you with a fast storage device, good for running disk-intensive programs or storing temporary files. But it has drawbacks: You must set it up and copy files to it each time you start your PC; it uses precious memory; and it goes *poof* when you turn off the power. A better way to put memory to use and speed up all disk operations is a *disk cache.*

Disk caches and RAM disks are often confused. A cache (pronounced *cash*) isn't a RAM disk at all. Instead, it's a large disk buffer—a holding place in memory for all information read from disk. The advantage comes when you want to read that same information again. Rather than fetching it from the disk a second time, a disk-caching program will read the information from the cache memory. The result is improved disk performance: You didn't have to reaccess the drive to get information already in the cache. Since the majority of disk operations involve this redundant type of reading, a cache will greatly improve your overall system speed.

Unlike a RAM disk, a cache contains a *copy* of information already saved on the hard drive. With a cache, you don't run the risk of losing data. The cache also saves you the time it takes to read that information, which is now read from fast memory instead of the slow disk. Information written to disk is written directly. Only read operations are sped up by the cache, so there is no possibility that information will ever be lost.

With all these benefits, you may wonder why you haven't been using a cache (if you're not already). Perhaps you didn't know about disk-caching, or didn't want to devote valuable memory to a cache. Another reason is that, before DOS 5, you had to buy a disk cache as a separate utility program. Finally, not all caches work with all hard drives and software. Because of the incompatibilities, a few people shy away from caches. This should all change with DOS 5's SMARTDrive caching device driver. Now, if you have the memory to spare, it's a good idea to put a disk cache to work and speed up your disk drives.

Using SMARTDRV.SYS with DOS 5.0

This section is about using the SMARTDrive disk-caching program, included with DOS version 5.0. If you haven't yet upgraded to DOS 5, skip ahead to the section "The Fastopen Command."

DOS 5.0 includes Microsoft's SMARTDRV.SYS cache device driver, which is similar to the caching program that comes with Microsoft Windows. Like the Windows version, DOS 5's SMARTDrive program has a few limitations: You can use SMARTDRV.SYS only on an 80286 or later microprocessor; you need a hard drive to get the most from SMARTDrive; and you need at least 512K of extended or 256K of expanded memory. That last item is important: you must have either extended or expanded memory to use SMARTDrive; there's no present way to use it in conventional memory only. (For PC/XT users and other users without extended or expanded memory, third-party cache programs exist. Check your local software vendor.)

SMARTDRV.SYS is installed using the device command in your CONFIG.SYS file. There are three optional parameters.

device=smartdrv.sys *initial minimum* /a

initial is the initial size of the caches, as measured in kilobytes. The minimum is 128K and the maximum is 8129K, or 8MB. Use a value that will leave you some extended or expanded memory if you need it. If *initial* isn't specified, the value 256 is assumed.

minimum is the minimum size of the cache, measured in kilobytes. SMARTDrive will adjust its size based on the requests of some applications. The value you specify for *minimum* is the smallest size to which SMARTDrive can be shrunk. You can specify values from 0 up through the value of *initial*. Note that values should be multiples of 16. If *minimum* isn't specified, the value 0 is used.

/a tells SMARTDRV.SYS to use expanded memory for the cache. If /a isn't specified, extended memory will be used.

Examples of defining a cache

Assume that SMARTDRV.SYS is in the \DOS directory on drive C. To define a 1024K cache with minimum size of 256K in extended memory, you would include the following command in CONFIG.SYS:

```
device=c:\dos\smartdrv.sys 1024 256
```

To define a 256K cache with a minimum size of 0 (the default) in expanded memory, you would include the following command in CONFIG.SYS:

```
device=c:\dos\smartdrv.sys /a
```

Unlike a RAM disk, a cache is not assigned a letter, nor is there any need to define more than one. If you need a larger cache, specify a large value for *initial*, always making sure you have enough memory left over for other uses.

> ❑ **NOTE:** *Do not use the Devicehigh command when loading SMARTdrive. Refer to the section "Using the Devicehigh Command" in Chapter 9 for more information on Devicehigh.*

Using a cache

Unlike a RAM disk, a cache improves the speed of all disk operations. And since the cache only keeps a copy of what's already on disk in memory, there's no danger of losing anything if the power goes off.

If you have a 386 or 486 computer and are using SMARTDrive with the /a switch, make sure your expanded memory manager, such as EMM-386.SYS, is installed *before* the line in CONFIG.SYS that starts the cache. (For a general rule on defining memory device drivers, refer to the latter part of Chapter 8.)

The only drawback to using a cache is that it uses memory. If you're running programs that require extended or expanded memory, don't be overly generous with the amount of memory you give the cache. A 128K cache is more then enough for even the most disk-intensive database operations. A 256K cache, which is SMARTDrive's default, is just about the perfect amount. And if you have enough memory, a RAM disk to speed up some operations and a cache to improve the rest makes a formidable supercharging combination.

The Fastopen Command

Since its debut in DOS version 3.3, the Fastopen command has created bewilderment among DOS owners. It has a confusing name and serves an unusual function. But you shouldn't be afraid of a command that can improve the speed of your system's file access.

Fastopen creates a cache, but not a disk cache. It simply keeps track of the file names and subdirectories you've accessed and their locations on disk. Once Fastopen has that information memorized, the speed of a second access to the same file or subdirectory is greatly improved.

The format of the Fastopen command is:

fastopen *drive=files /x*

drive is a disk drive letter, followed by a colon. It indicates the hard drive you want Fastopen to monitor.

files is the number of subdirectories and file names you wish Fastopen to keep track of. The default value is 48, although you can specify any number up to 999. Each file will eat up about 50 bytes of memory.

/x tells DOS to keep Fastopen's buffer in expanded memory (if you have it). Otherwise, Fastopen's buffer is stored in conventional RAM.

To take advantage of Fastopen, you can put a line in your AUTOEXEC.BAT file such as:

```
fastopen c:=30
```

The above line loads Fastopen into memory and tells it to save the most recent 30 files accessed on drive C. Any additional access to those files will be much quicker now that Fastopen is installed. Access to the file's contents will not be faster; Fastopen is only a fast file *opener* (hence its name). To speed up disk access to a file's contents, you must use a true disk cache (which will then make Fastopen redundant).

CHAPTER

11

Display It Your Way

You probably stare at the screen a lot. If you're not already intimately familiar with it, you will be. Except for the occasional beep, DOS can communicate with you only by displaying something. Anything you can do to improve this communication shows up right away.

The basic elements of your display are fixed: its size, shape, and resolution; whether it can display color and graphics; and its screen color, if it's a monochrome monitor. But you can still make your screen more communicative and easier to read. This chapter shows you how to:

- Make DOS display more than the usual number of lines
- Make the system prompt both useful and attractive
- Control the appearance of the cursor
- Blank the screen
- Change the foreground and background colors if you have a color monitor

Asking DOS for More Lines

If you're using version 4 or later of DOS and your system has an EGA or VGA display, you can use the Mode command to tell DOS to display more than the usual number of lines. This can be quite helpful when you're displaying long files or lists of directory entries. You can also set the number of columns on the display to 40 or 80 on any color monitor.

To display more lines, your display must be either an EGA or VGA, and your CONFIG.SYS file must contain a Device command that tells DOS where to find ANSI.SYS.

To fit more lines on the screen, the characters must, of course, be smaller than the normal characters. Depending on the size of your display, you may find that the characters are too small to read comfortably for any length of time; you'll also find that the display scrolls more slowly.

This form of the Mode command has three parameters:

mode con cols=*columns* lines=*rows*

con is the name of the console device (the display). It can be followed by an optional colon.

cols=*columns* specifies the number of columns or characters per line on the display, either 40 (wide) or 80 (normal). For example, to switch over to the wide display, enter:

```
C>mode con cols=40
```

DOS should clear the screen and display the system prompt in fat, double-wide letters. You'll notice that many commands, including Dir and Debug, are well suited to a wide display. To switch back to 80 characters per line, enter:

```
C>mode con cols=80
```

lines=*rows* specifies the number of lines that will be displayed. If your display is an EGA, *rows* can be 25 or 43; if your display is a VGA, *rows* can be 25, 43, or 50.

To display 43 lines, enter:

```
C>mode con lines=43
```

DOS should clear the screen and display the system prompt in smaller letters. Now enter a Directory command to see the difference. The change remains in effect until you change the number of lines with another Mode command or restart your computer.

If DOS replies *Function not supported on this computer*, you don't have an EGA or VGA display. If DOS replies *Invalid parameter*, you aren't using the right version of DOS. Versions prior to DOS 4 supported changing the column width with the mode *display* syntax. If DOS replies *ANSI.SYS must be installed to perform requested function*, put a Device command in CONFIG.SYS that tells DOS where to find ANSI.SYS, restart your computer, and enter the Mode command again.

To return the display to the standard number of lines, enter:

```
C>mode con lines=25
```

Sprucing Up the System Prompt

The Prompt command lets you define a system prompt as spartan or baroque as you like. But keep two things in mind as you tailor the prompt: You want to find the information at a glance, not hunt for it, so the system prompt should be informative but not too busy. And resist the temptation to make the prompt cute—you'll read it hundreds or even thousands of times, and something terribly clever gets old fast.

For demonstration purposes, this chapter defines a system prompt that tells you three things (each is followed by the special code that the Prompt command uses to represent it).

- The time (*$t*)
- The date (*$d*)
- The current drive and directory (*$p*)

The following examples treat current drive and directory more prominently than time and date because you'll need that information more often.

Some Sample System Prompts

Before you try these examples, make sure ANSI.SYS is on your system disk. Also, CONFIG.SYS must contain a Device command that tells DOS where to find ANSI.SYS. Here, the current directory is \BATCH on drive C; if you have a directory with that name, change to it. Otherwise, substitute your current directory name for \BATCH in the examples.

To start designing a system prompt, enter the following Prompt command to display the time, the date, and the current drive and directory:

```
C>prompt $t$d$p
```

DOS displays the following prompt (the time and date will vary):

```
10:39:09.54Fri 10-16-1992C:\BATCH_
```

That's a pretty rough beginning. Everything runs together, making it hard to pick out the information you want. The time is too detailed—even at a track meet, time is kept only to the nearest tenth of a second if a human being holds the stopwatch. You've lost 36 characters from the command line; the cursor is almost in the middle of the screen. Not only that, the prompt runs right into the command you enter. For example, if you entered *dir *.com ¦ sort*, the command line would look like this:

```
10:39:34.15Fri 10-16-1992C:\BATCHdir *.com ¦ sort
```

This certainly isn't an ideal solution. (The Macintosh users would be rolling in the aisles!) But you can correct several problems easily:

- Make the time easier to read by putting three *$h* (backspace) characters after the *$t* to erase the hundredths of a second.
- Break the prompt into three lines with *$_* (carriage return), placing the time, the date, and the current drive and directory on separate lines.
- Separate what is to be typed from the prompt by including a space after the last character in the prompt definition.

Enter *prompt* to change the system prompt to the more manageable C>, then enter the following Prompt command:

```
C>prompt $t$h$h$h$_$d$_$p<space>
```

The following table describes the elements of this Prompt command:

Element	***Description***
$t	Displays the time
hh$h	Backspaces three times, erasing the hundredths of a second
$_	Moves to the next line
$d	Displays the date
$_	Moves to the next line
$p	Displays the current drive and directory name
<space>	Separates the system prompt from the command you type

Now the system prompt is much easier to read:

```
10:41:58
Fri 10-16-1992
C:\BATCH _
```

You can find information you need more quickly, it's obvious where a command will appear when you type it, and you're only giving up nine characters of the command line. (If you want to get rid of the seconds, too, you could put six backspaces—*hhhhhh*—instead of three in the Prompt command.)

But this system prompt takes three lines; to see how fast it eats up space on the screen, press the Enter key a few times. You can solve that without giving up any of the information.

You'll need to know the name of the current drive and directory far more often than the time or date, but it's still handy to have the time and date on the screen. Why not display the time and date in a fixed location—the upper right corner of the screen is nicely visible, yet out of the way—and repeat the current drive and directory with each system prompt?

The ANSI.SYS commands for saving the cursor position, moving the cursor, and restoring the saved cursor position let you do this. Instead of using *$_* to move to the next line, use two ANSI.SYS Move Cursor commands to display the time and date in the upper right corner of your screen. You can keep track of the command-line position by saving the cursor position before you display the time and date, and by restoring the cursor position before you display the current drive and directory. (See Appendix A for more information about ANSI.SYS commands.)

Enter the following Prompt command. Because it's quite long, each part of the command is shown here on a separate line, but enter it as a single command on a single line. (Don't enter the explanations listed at the right.)

```
prompt
$e[s                Saves the current cursor position
$e[1;67H            Moves the cursor to row 1, column 67
$d                  Displays the date
$e[2;67H            Moves the cursor to row 2, column 67
$t                  Displays the time
$h$h$h              Backspaces three times, erasing the hundredths of a second
$e[u                Restores the saved cursor position
$p                  Displays the current drive and directory name
<space>             Separates the system prompt from the command you type
```

You're going to reuse this Prompt command in a just a moment, so don't type anything after you have pressed the Enter key; don't even press the Enter key again.

Now DOS displays the system prompt in two different parts of the screen. The time and date are in the upper right corner of the screen, starting at column 67 in rows 1 and 2, and the current drive and directory are displayed on the command line:

```
                                                    Fri 10-16-1992
                                                    10:50:07

C:\BATCH _
```

Note that if DOS displays only part of the prompt and responds *Out of environment space*, there isn't enough room in the environment for a system-prompt definition this long (44 bytes). As Chapter 8 explained, there's a limit to the size of the environment in which DOS stores the system-prompt definition. If you haven't put a Shell command in CONFIG.SYS to expand the environment, take the time now to read "Increasing the Size of the Environment" in Chapter 8, then come back and finish this chapter.

This two-part system prompt meets the requirements discussed previously, but there's one last change: If you display the current drive and directory in reverse video, they'll stand out even better. This requires adding a couple of ANSI.SYS commands to the Prompt command. Because the changes occur close to the end of the command, however, you don't have to retype the whole thing; press F3 and DOS should redisplay the last Prompt command you entered:

```
                                                    Fri 10-16-1992
                                                    10:55:03
C:\BATCH prompt $e[s$e[1;67H$d$e[2;67H$t$h$h$h$e[u$p _
```

If you haven't used the editing keys before to reuse the last DOS command, refer to Chapter 13, "Increasing Your Keyboard's IQ."

Press the Backspace key three times; this should erase the space and *$p*:

```
                                                Fri 10-16-1992
                                                10:55:30
C:\BATCH prompt $e[s$e[1;67H$d$e[2;67H$t$h$h$h$e[u_
```

You're going to finish the Prompt command with an ANSI.SYS Set Attribute command to start reverse video, *$p* to display the current drive and directory, another ANSI.SYS Set Attribute command to turn off all display attributes, and a space. (Don't enter the explanations listed at the right.)

`$e[7m`	Starts reverse video
`$p`	Displays the current drive and directory name
`$e[m`	Restores normal display attributes
`<space>`	Separates the system prompt from the command you type

Type the new characters at the end of the previous command:

```
                                                Fri 10-16-1992
                                                10:56:57
C:\BATCH prompt $e[s$e[1;67H$d$e[2;67H$t$h$h$h$e[u$e[7m$p$e[m<space>
```

When you press Enter the current drive and directory really stand out:

```
                                                Fri 10-16-1992
                                                10:57:18
C:\BATCH
```

This satisfies the most important goals of a system prompt: It's informative, direct, and easy to read. It doesn't use up screen space too quickly, because only the current drive and directory are displayed on the command line. If you like this system prompt, take a few moments now to put it in a batch file while it's still fresh in your mind. Later you can copy it to AUTOEXEC.BAT, to the batch file that restores your system prompt (call it PROMPTRS.BAT), or wherever else you might want it.

Spicing Up the System Prompt with a Bit of Color

Color can be overused to disadvantage, and you might be bumping up against the size limit of the environment, but if you're using a color display you can add a couple of ANSI.SYS commands to polish your system prompt. The form of the prompt is unchanged, but the time and date are

displayed in cyan, and the current drive and directory are displayed in high-intensity yellow on a red background.

Again, the Prompt command is shown broken down into its elements, but you must enter it as a single command. (Don't enter the explanations listed at the right.)

`prompt`	
`$e[s`	Saves the current cursor position
`$e[1;67H`	Moves the cursor to row 1, column 67
`$e[36m`	Sets the foreground color to cyan
`$d`	Displays the date
`$e[2;67H`	Moves the cursor to row 2, column 67
`$t`	Displays the time
`$h$h$h`	Backspaces three times, erasing the hundredths of a second
`$e[u`	Restores the saved cursor position
`$e[1;33;41m`	Sets the foreground to high-intensity yellow and the background to red
`$p`	Displays the current drive and directory name
`$e[m`	Restores normal display attributes
`<space>`	Separates the prompt from the command you type

This system prompt has the same form as the previous one, but now it's displayed in color. The relative boldness of the colors suits the importance of what they display. You might like other colors better; try some different combinations by changing the ANSI.SYS commands. When you find an effect you like, save the Prompt command in a batch file.

Enough about the Prompt command. In addition to the system prompt, what else is always on the screen? The cursor, of course.

Don't Curse the Cursor

Usually the cursor is an underline, but this is established by software, not hardware. In some application programs the cursor fills the entire character space or half the character space or disappears altogether; some programs use several types of cursors. The three short programs described here give you the same sort of control over the cursor. They work with all models of the IBM PC and IBM-compatible machines.

The following programs don't affect the cursor's blinking, by the way; that *is* fixed in the hardware.

Creating the programs

You create these programs with Debug, using the procedure described in Chapter 5. If you haven't yet gone through the examples in Chapter 5, you might do so before continuing here.

The description of each program includes two figures. The first shows the contents of the Debug script file you create, and the second shows how DOS should respond when you redirect the input of the Debug command to the script file. Proofread the script files carefully. Pay particular attention to the blank line that follows the program statements, and make sure that the last line is *q*; if there is a mistake in either of these lines, you'll have to restart DOS when you try to create the COM file.

If it doesn't work

If nothing seems to happen and the keyboard refuses to respond when you enter the Debug command, restart DOS. Load the script file into your text editor or word processor, and proofread it carefully against the figure that shows its contents. After correcting any errors you might have found, reenter the Debug command.

If DOS displays a series of commands and responses after you enter the Debug command, proofread the display carefully against the figure that shows the DOS response to creating the program. If the figure and the text on your screen don't agree, go back and edit the script file again. Proofread it carefully against the figure that shows the contents of the script file, then save it and enter the Debug command again.

Would you prefer a block cursor?

Some computers use a blinking cursor that fills the entire character space. BIGCURS.COM is a program that changes the cursor to a solid block. You can change the cursor back to normal with NORMCURS.COM, described next.

Use your text editor, word processor, Editor, or Edlin to create a file named BIGCURS.SCR and enter the lines shown in Figure 11-1. (The line numbers are for reference only.)

```
 1:  a 100
 2:  mov ah,1
 3:  mov cx,10c
 4:  int 10
 5:  int 20
 6:
 7:  r cx
 8:  9
 9:  n bigcurs.com
10:  w
11:  q
```

Figure 11-1. *BIGCURS.SCR: Making the cursor a block.*

Make certain that line 6 of BIGCURS.SCR is blank and that the last line is a *q*. When the file is correct, save it and then enter the following Debug command:

```
C>debug < bigcurs.scr
```

DOS should respond as shown in Figure 11-2.

```
-a 100
xxxx:0100 mov ah,1
xxxx:0102 mov cx,10c
xxxx:0105 int 10
xxxx:0107 int 20
xxxx:0109
-r cx
CX 0000
:9
-n bigcurs.com
-w
Writing 0009 bytes
-q
```

Figure 11-2. *DOS response to creating BIGCURS.COM.*

If your screen doesn't agree with Figure 11-2, see the section "If it doesn't work" for instructions. If your screen agrees with Figure 11-2, enter *bigcurs*; the cursor should change to a flashing block.

Restoring the normal cursor

Sometimes when an application program changes the shape of the cursor, the cursor remains a hyphen or a solid block when the application returns to DOS. You can usually clear the screen and restore the normal cursor with a Mode command; enter *mode con cols=80*, *mode co80*, or *mode mono*, depending on what version of DOS and what sort of monitor you're using. NORMCURS.COM is a short program that also restores the cursor, but more quickly and without clearing the screen.

Using your text editor, word processor, Editor, or Edlin, create a file named NORMCURS.SCR and enter the lines shown in Figure 11-3.

Make certain that line 12 of NORMCURS.SCR is blank and that the last line is a *q*. When you're sure the file is correct, save it and enter the following Debug command:

```
C>debug < normcurs.scr
```

DOS should respond as shown in Figure 11-4.

```
 1:  a 100
 2:  mov ah,0f
 3:  int 10
 4:  cmp al,7
 5:  jz 10d
 6:  mov cx,607
 7:  jmp 110
 8:  mov cx,0b0c
 9:  mov ah,1
10:  int 10
11:  int 20
12:
13:  r cx
14:  16
15:  n normcurs.com
16:  w
17:  q
```

Figure 11-3. *NORMCURS.SCR: Restoring the normal cursor.*

```
-a 100
xxxx:0100 mov ah,0f
xxxx:0102 int 10
xxxx:0104 cmp al,7
xxxx:0106 jz 10d
xxxx:0108 mov cx,607
xxxx:010B jmp 110
xxxx:010D mov cx,0b0c
xxxx:0110 mov ah,1
xxxx:0112 int 10
xxxx:0114 int 20
xxxx:0116
-r cx
CX 0000
:16
-n normcurs.com
-w
Writing 0016 bytes
-q
```

Figure 11-4. *DOS response to creating NORMCURS.COM.*

If your screen doesn't agree with Figure 11-4, see the section "If it doesn't work" for instructions. If your screen agrees with Figure 11-4, enter *normcurs*. The block cursor should revert to an underline (the normal cursor).

Making the cursor invisible

For those times when you don't want a cursor, NOCURS.COM makes it disappear. Only temporarily, of course; NORMCURS.COM or BIGCURS.COM will bring it back.

The next section, for example, describes a batch file that blanks your display screen; the batch file uses NOCURS.COM to make the cursor disappear and NORMCURS.COM to restore it. Another time you might not want a cursor (strictly for aesthetic reasons) is when a menu is displayed.

Use your text editor, word processor, Editor, or Edlin to create a file named NOCURS.SCR and type the lines shown in Figure 11-5.

```
1:  a 100
2:  mov ah,1
3:  mov ch,20
4:  int 10
5:  int 20
6:
7:  r cx
8:  8
9:  n nocurs.com
10: w
11: q
```

Figure 11-5. *NOCURS.SCR: Making the cursor invisible.*

Make certain that line 6 of NOCURS.SCR is blank and that the last line is a *q*. When the file is correct, save it and then enter the following Debug command:

```
C>debug < nocurs.scr
```

DOS should respond as shown in Figure 11-6.

```
-a 100
xxxx:0100 mov ah,1
xxxx:0102 mov ch,20
xxxx:0104 int 10
xxxx:0106 int 20
xxxx:0108
-r cx
CX 0000
:8
```

Figure 11-6. *DOS response to creating NOCURS.COM.* *(continued)*

Figure 11-6. *continued*

```
-n nocurs.com
-w
Writing 0008 bytes
-q
```

If your screen doesn't agree with Figure 11-6, see the section "If it doesn't work" for instructions. If your screen agrees with Figure 11-6, enter *nocurs*; the cursor should disappear. Now enter *bigcurs* to display the block cursor. Finally, enter *normcurs* to restore the normal cursor.

The next section describes a batch file that gives you your first chance to put these programs to use.

Make the World Go Away

When you're going to leave the computer for a few hours but don't want to turn it off, you should make the screen blank. You could turn down the intensity on your monitor, or even turn the monitor off, but it's easy to use a batch file, especially if it's as simple as a menu selection.

BLANK.BAT blanks the screen, then restores the display at the touch of a key. It requires three programs that you have created: NOCURS.COM and NORMCURS.COM, described earlier in this chapter, and REPLY.COM, which is described in Chapter 7. Figure 11-7 shows the commands in BLANK.BAT.

```
1:  @echo off
2:  cls
3:  nocurs
4:  reply
5:  normcurs
```

Figure 11-7. *BLANK.BAT: Making the screen blank.*

BLANK.BAT is straightforward:

- Line 2 clears the screen.
- Line 3 turns the cursor off.
- Line 4 waits for you to press a key.
- Line 5 restores the cursor no matter what key you press.

To test BLANK.BAT, enter *blank*. The screen should go blank. Pressing any key should restore the display.

If you assign the BLANK.BAT command to a key—for example, [0;48;"blank";13p, which assigns it to Alt-B—you can blank your screen by pressing Alt-B at the DOS prompt. See "Assigning a Command to a Key" in Chapter 3 for a description of how to assign a command to a key. If there's a chance that someone might see the blank screen and, thinking that you forgot, turn your system off, you might put a note on the keyboard warning colleagues against that sort of help.

Discouraging Casual Use with BLANK.BAT

You can add a limited form of security to BLANK.BAT by requiring a specific key to be pressed before the display is restored. It's certainly not very secure—Ctrl-C ends the batch file and returns to DOS—but it can discourage casual use or snooping.

Figure 11-8 shows a revised version of the batch file BLANK.BAT called BLANKSEC.BAT that requires you to press Ctrl-Home (key code 0;119) to restore the screen. To substitute another key, replace *120* in line 6 with the key code of the special key plus 1, and *119* in line 7 with the key code of the special key. See Appendix D for a list of special key codes.

```
1:     @echo off
2:     cls
3:     nocurs
4:  :GET_REPLY
5:     reply
6:     if errorlevel 120 goto GET_REPLY
7:     if not errorlevel 119 goto GET_REPLY
8:     normcurs
```

Figure 11-8. *BLANKSEC.BAT: Requiring a specific key to restore the screen.*

BLANKSEC.BAT is much the same as BLANK.BAT:

- Line 2 clears the screen.
- Line 3 turns the cursor off.
- Line 5 waits for you to press a key.
- Line 6 goes back to wait for another key if the key code of the key pressed is greater than the special key code.
- Line 7 goes back to wait for another key if the key code of the key pressed is less than the special key code.
- Line 8 restores the cursor. Because of the tests in lines 6 and 7, the batch file gets to line 8 only if you press the special key.

To test BLANKSEC.BAT, type *blanksec*. The screen should go blank, but pressing any key other than the special key has no effect. Press Ctrl-Home to restore the cursor.

Changing the Screen Colors

The remainder of this chapter deals with changing the foreground and background colors. If you aren't using a color display, you can go on to Chapter 12, "Print It Your Way."

A color display normally displays white characters on a dark screen; in the parlance of ANSI.SYS, this is a white *foreground* on a black *background*. As shown in Chapter 3, you can change these colors with the ANSI.SYS Set Attribute command. Appendix A summarizes all the ANSI.SYS commands; Figure A-3 includes a table of all the foreground and background color codes used by the Set Attribute command.

As in Chapter 3, the examples here use the Prompt command to change the foreground and background colors.

Changing the foreground color

The most common colors used by monochrome monitors to display their wares are green, amber, and a pale blue that's almost white. All three have their adherents, who make contradictory claims about the efficiency or restfulness of various colors; you can make your color display emulate any of these simply by changing the foreground color.

For example, a common color used by monochrome displays is green. You can have a green display with one command:

```
C>prompt $e[32m
```

To see the effect of this and the remaining commands in this section, clear the screen and display the directory after changing the color.

Amber is highly touted as more restful than other colors. You can approximate amber with yellow; change the 32 (green foreground) in the preceding command to 33 (yellow foreground):

```
C>prompt $e[33m
```

The appearance of yellow on different color displays varies more than any other color. Adjusting the brightness and contrast controls may help. If the characters look more red than yellow (or amber), you may have to specify high-intensity yellow with the following command.

```
C>prompt $e[1;33m
```

If you specified high-intensity yellow, enter the following Prompt command to turn off all attributes and restore the normal white letters on a black background:

```
C>prompt $e[m
```

Finally, the cyan foreground color comes close to the pale blue used by a few displays:

```
C>prompt $e[36m
```

Again, high-intensity cyan may come closer to the color you want; the command for high-intensity cyan is *prompt $e[1;36m*.

There are three more colors you can try, if you wish: red (31), blue (34), and magenta (35).

Changing the background color

You aren't limited to a black background. You can make the background any of the eight (counting black and white) colors. For example, enter the following command to change the background to blue:

```
C>prompt $e[1;44m
```

Nothing seems to have changed; there isn't even a system prompt to show the new color because the system prompt consists solely of the command to change the background color. You have to clear the screen to see the new background color:

```
C>cls
```

There's your blue background. Keep in mind this need to clear the screen if you decide to change your system's colors.

Changing both foreground and background colors

Not all combinations of foreground and background colors are useful. Green characters are invisible on a green background, of course, and nearly invisible on a cyan background. Other color combinations, such as high-intensity red foreground on a blue background, are quite distinct, but you wouldn't want to look at the screen for very long (unless your sanity isn't important to you).

The best way to get a feeling for what you like is to try several color combinations. You can check them out pretty quickly by using a Prompt command to set the colors, then clearing the screen and displaying the directory. Figure 11-9 shows some combinations with sufficient contrast to read. Appendix A lists all the available colors and their numbers.

Other combinations are possible, of course. Because color preference is highly subjective, experiment for a while—find something to match your office decor or your mood for the day. The high-intensity foreground colors provide better contrast, but you might like normal intensity better. Try several combinations.

If you prefer a dark foreground on a light background, try the combinations shown in Figure 11-10 and any others that you think you might like.

Defining your own standard colors

You can define your own standard colors by putting an Echo command in AUTOEXEC.BAT that sends the appropriate ANSI.SYS command to the console. For example, if you wanted to set your standard display colors to a high-intensity yellow foreground on a blue background (popular in some southern European countries), you would put *echo* ␛*[1;33;44m* in AUTOEXEC.BAT. But there's a bit more to it than that, unfortunately. If you don't plan to define your own standard colors, you can skip the next few sections and go on to Chapter 12.

Command	*Foreground*	*Background*
prompt $e[37;41m	White	Red
prompt $e[37;44m	White	Blue
prompt $e[1;33;41m	High-intensity yellow	Red
prompt $e[1;33;44m	High-intensity yellow	Blue
prompt $e[1;36;44m	High-intensity cyan	Blue
prompt $e[1;37;44m	High-intensity white	Blue
prompt $e[1;37;46m	High-intensity white	Cyan

Figure 11-9. *Foreground-background color combinations.*

Command	*Foreground*	*Background*
prompt $e[30;47m	Black	White
prompt $e[31;47m	Red	White
prompt $e[34;47m	Blue	White
prompt $e[30;46m	Black	Cyan
prompt $e[31;46m	Red	Cyan
prompt $e[34;46m	Blue	Cyan

Figure 11-10. *Color combinations with a light background.*

Changing colors can complicate your life a bit

If, in AUTOEXEC.BAT, you define a standard set of colors *and* you define a system prompt that sets display attributes, you'll never see your standard colors unless the prompt definition ends by setting the standard display colors. DOS displays the system prompt as soon as AUTOEXEC.BAT finishes, and the last display attributes set in the prompt definition are the ones that DOS uses.

For example, you typed a Prompt command example earlier in this chapter that displayed the current drive and directory in reverse video and ended with *$e[m*, which turned off all the display attributes. Each time DOS displayed that system prompt, the screen would be set to a white foreground on a black background, even if you had defined a different set of colors in AUTOEXEC.BAT.

So your Prompt command must not only define the system prompt, its last act must be to set your standard colors. If you wanted your screen to be blue characters on a cyan background (popular in the northern latitudes), your Prompt command would have to end with *$e[34;46m*.

Furthermore, if you use BLANK.BAT, it must start by changing the background color to black (insert *echo* [E]*[m* after the first line) because all the Clear Screen command does is erase foreground characters. Only if the background color is black will the screen be blank; if the background is any other color, the screen will be filled with that color. If BLANK.BAT changes the background color to black, it, too, must restore your background color before it returns to DOS.

Watch Out for Those Batch Files

Any batch file that sets display attributes with an ANSI.SYS command must restore your standard colors before it returns to DOS. You might use several batch files that set display attributes; this book is filled with them. All it takes is one [E]*[7m* to turn your display into black characters on a white background or [E]*[m* to set what DOS considers a normal display (white characters on a black background). You'll also find that some of these batch files look fine when the normal attribute is set but need some revision if different foreground or background colors are set.

Keeping track of all the places where you must restore your colors can be a chore. And should you decide to change your standard foreground and background colors, you'll have to change every one of the batch files that restores the standard colors.

Instead, create a batch file that includes a Prompt command that restores the system prompt and ends by setting the standard colors, followed

by an Echo command that sets the standard colors, and then a Clear Screen command. Call this batch file PROMPTRS.BAT, and use it as the last command in any batch file that changes either the prompt or the display attributes. (It was suggested previously that you create a batch file named PROMPTRS.BAT to restore your system prompt; if you did, modify it to match this description.)

For example, suppose you defined your standard colors as a black foreground on a cyan background, and your system prompt consisted of the current drive and path displayed in reverse video. Figure 11-11 shows the commands PROMPTRS.BAT would contain.

```
1:  @echo off
2:  prompt $e[7m$p$e[30;46m
3:  echo ←[0;30;46m
4:  cls
```

Figure 11-11. *PROMPTRS.BAT: Restoring the prompt and standard colors.*

The attributes *30;46* in line 2 cause DOS to set the background and foreground colors each time it displays the system prompt. In line 3, the 0 attribute turns off any attributes that may be set—such as different colors, reverse video, or high intensity—and the attributes *30;46* (again!) set the colors immediately. (Remember, the system prompt isn't displayed until the batch file returns to DOS, so line 2 has a delayed effect.) The *cls* command in line 4 fills the screen immediately with the background color.

Make *promptrs* the last command in any batch file that sets a display attribute or changes the system prompt, including AUTOEXEC.BAT. If you get tired of black on cyan as your colors, just change the attributes in lines 2 and 3 of PROMPTRS.BAT.

Application Programs Get into the Act

Some application programs define their own foreground and background colors. They restore any colors you have defined when they return to DOS, but the screen remains black until you fill it. This is only a superficial blemish, but if it bothers you, you can avoid it by running the application program with a two-line batch file: the command that starts the application program followed by *cls*.

It's worth the effort to define your own colors if you discover a combination that you prefer. Just take the time to create PROMPTRS.BAT, and you can manage the situation with a minimum of fuss.

CHAPTER

12

Print It Your Way

The printer is a key part of your computer system. Whatever types of application programs you use—word-processing, spreadsheet, database, graphics, or something more specialized—the printer produces the documents that record the planning, progress, and completion of your work. Your printer may even remind you how much you use the computer when you run out of paper and discover that box of 3000 sheets is gone.

The first dot-matrix printers offered by IBM were manufactured by Epson. IBM now makes its own printers, but they still use basically the same commands as those early Epson machines with IBM labels. Because of the size of the Epson-compatible market, most printer manufacturers make dot-matrix printers that use the same commands and character set. Similarly, Hewlett-Packard's LaserJet printer became a de facto standard due to its early success. Though IBM now makes its own laser printer, the LaserJet remains the standard for DOS laser printers.

There are exceptions, of course. Other laser printers and specialty printers use different commands, and some dot-matrix printers don't follow the IBM-Epson conventions. If you're buying a new printer, make sure it's compatible and ask for a demonstration if there's some question. If you have a printer, check the manual to see whether the printer follows the industry standard. There should be a chart or charts that show you whether the printer can print the IBM extended character set.

Some manufacturers, including even Epson and Hewlett-Packard, have decided that it isn't necessary to maintain strict compatibility. Check your printer's manual to see whether your printer uses the same commands described in this book.

Competition in the printer market has lowered the cost of dot-matrix printers while increasing the quality. Figure 12-1 illustrates some of the capabilities of today's dot-matrix printers. Chapter 4 showed you how to tap some of your printer's capabilities; this chapter shows you how to apply these capabilities to some real-life situations.

Laser printer prices only recently dropped below the $1,000 mark, and now even cheaper systems are offered. Laser printers give you advanced graphics and speed, and they're quiet. They're preferred for high-quality business output and are a necessity for graphics and desktop publishing. Figure 12-2 is a sample of a laser printer's output.

Printer Commands

As shown in Chapter 4, most Epson printer commands are one, two, or three bytes long, and HP PCL commands are usually much longer.

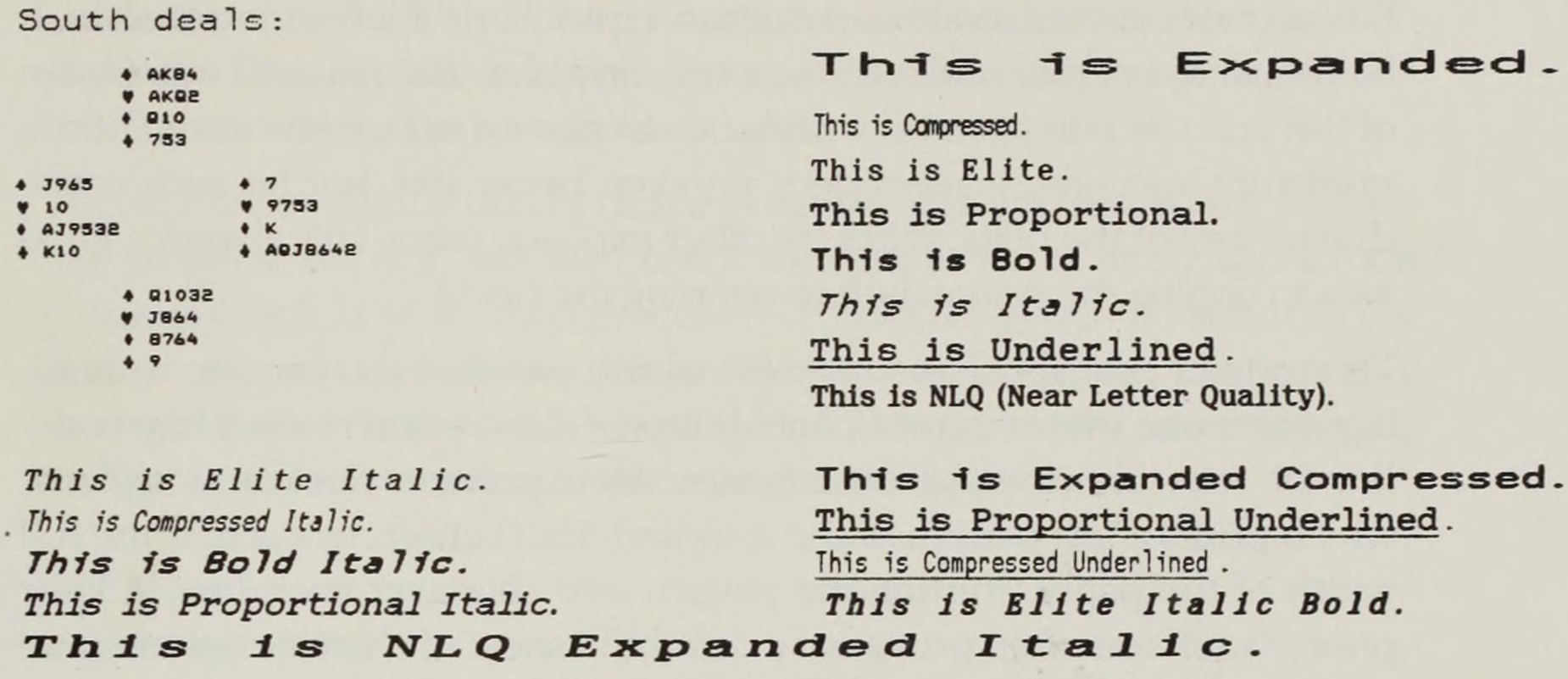


Figure 12-1. *Sample of dot-matrix printer output.*

Although most parameters are fixed (such as Alt-15 or the letter *D*), some printer commands require a parameter whose value can vary, such as the commands that set the page depth or the left margin or the commands on a laser printer to change the font or appearance of text.

There are four ways to send commands to the printer. You can redirect the output of an Echo command, copy a file of commands to the printer, include printer commands in a file to be printed, or run a program that sends commands to the printer.

The examples in Chapter 4 used several printer commands, and this chapter uses several more. Appendix B is a quick reference to the more common printer commands, but the manual that came with your printer is the complete reference. (The manual may use abbreviations such as SO or DC1 to refer to characters whose code is less than 31; Figure C-4 in Appendix C explains these abbreviations and shows which character each represents.)

This is a long chapter. It discusses operating and caring for your printer, explains some of the performance features added to the Print command in version 3 of DOS, and then shows several batch files and programs that let you use some of your printer's capabilities. It concludes with a batch file that creates a Printer Setup menu you can use to control the type of print and some of the formatting capabilities built into your printer. This batch file is included in the menu system described in Chapter 15.

Operating Your Printer

In addition to turning your printer on and off, adding paper, and changing the ribbon or toner, you should clean it once in a while. Dust and small

bits of paper gather inside and can cause problems if left to accumulate. It helps just to open or remove any covers and blow the material out. (A can of "air" can be purchased at your local electronics store.) And one of those miniature vacuum cleaners does an even better job; just be sure not to disturb any of the belts, wires, or other exposed parts. (It's always a good idea to unplug the system before opening the case.)

Your printer probably has some special features that you can use by holding down one or more front-panel buttons; some printers even have one-line or two-line displays with menus. Most printers can run a *self test,* which prints a test pattern of the standard ASCII characters across the full width of the paper, shifting the pattern one character each line. A laser printer's self test often prints out a delightful sheet of characters, fonts, and graphics to show what it can do. The self test of a laser printer is shown in Figure 12-2. It's quite a complete rundown of what the printer is capable of, as well as being informative.

```
 !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_'abcdefghijklmno
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_'abcdefghijklmnop
"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_'abcdefghijklmnopq
#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_'abcdefghijklmnopqr
$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_'abcdefghijklmnopqrs
%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_'abcdefghijklmnopqrst
&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_'abcdefghijklmnopqrstu
```

Figure 12-2. *Laser printer self test.*

The self test of a dot-matrix printer—a few lines are shown in Figure 12-3—continues for 15 minutes or so; you can stop it sooner by turning the printer off.

```
 !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnop
"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopq
#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqr
$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrs
```

Figure 12-3. *Dot-matrix printer self test.*

You can run the self test even if the computer is turned off or the printer isn't attached to a computer; it's a good way to make sure that you installed a new ribbon correctly or to make sure the printer works. To run the self test on an Epson or on most IBM printers, hold down the Form Feed button while you turn the printer on. On the LaserJet III, run a self test by pressing the Test button on the printer's front panel. Check your printer's manual to find instructions for running the self test.

Some dot-matrix printers have a feature that causes the printer to print the two-digit hexadecimal number that represents each character it receives,

rather than printing the character itself. For reasons described under "Displaying a File with the Dump Command" in Chapter 5, you're apt to see this feature referred to as *hex dump mode*.

This feature is usually used to test a program or batch file that isn't behaving properly, but you can also use it to see exactly what an application program (such as your word processor or graphics program) sends to the printer. Figure 12-4 shows three lines—containing the characters 0 through 9—as they would be printed, followed by the corresponding lines of hexadecimal data an Epson printer would produce.

```
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9

0F 30 20 31 20 32 20 33 20 34 20 35 20 36 20 37 20 38 20 39
0D 0A 1B 34 30 20 31 20 32 20 33 20 34 20 35 20 36 20 37 20
38 20 39 1B 35 12 0D 0A 1B 45 30 20 31 20 32 20 33 20 34 20
```

Figure 12-4. *Printer output in ASCII and hexadecimal.*

In the hexadecimal portion of Figure 12-4, *30* through *39* are the hexadecimal codes for the ASCII characters 0 through 9, *20* is a space, *0D* is a carriage return, and *0A* is a linefeed. The remaining characters are printer commands. The *0F* in the first byte of the first row is character 15, or the Start Compressed Print command. *1B* is the Escape character, so *1B 34* starting in the third byte of the second row is ␛4 (the Start Italics command). Starting with the fourth byte of the third row, *1B 35* is ␛5 (the Stop Italics command), *12* (character 18) is the Stop Compressed Print command, *0D 0A* is the carriage return and linefeed for the second line, and *1B 45* is ␛E (the Start Bold Print command).

To use this feature on an Epson or on most IBM printers, hold down both the Line Feed and Form Feed buttons when you turn the printer on. If you have a different make of printer, check the manual to see whether it offers this feature and, if so, how to start it.

Fine-Tuning the Print Command

This section doesn't describe how to use the Print command, but rather suggests how you might optimize its performance. The Print command is covered in the DOS manual and *Running MS-DOS*. If you don't use the Print command very often or if you're satisfied with how it works, you can safely skip this section and go on to "Printing a File in Pages," which describes the first batch file of this chapter.

The computer can do only one thing at a time, so the Print program doesn't really print a file simultaneously—it takes turns with whatever else the computer is doing. But the PC's clock ticks every 55 milliseconds (55 thousandths of a second), and every time it ticks DOS can switch jobs, so DOS has a chance to switch jobs 18.18 times each second. Things can change so quickly that it seems as if two things are happening at the same time.

The 55-millisecond interval between clock ticks, called a *tick,* is the measure of work time that DOS uses. Beginning with version 3 of DOS, the Print command lets you control how many ticks DOS will work before giving Print a chance and how many ticks Print will work before giving control back to DOS. You can also control how much memory the Print program uses to store what will be printed; this area of memory is called the *print buffer*.

> ❑ **NOTE:** *The printer has some memory of its own that it uses to store the data sent to it; this memory is usually called the* printer buffer, *raising the distinct possibility for confusion. Remember, you can control the size of the* print *buffer, but the* printer *buffer is part of the printer's hardware.*

Here, then, are the factors that interact to affect Print command performance:

- The size of the print buffer. You can control this with the /B parameter of the Print command. If you don't specify otherwise, DOS assigns 512 bytes to the print buffer. Increasing /B means that the Print program uses the disk less often, increasing its speed, but it also decreases the amount of memory available to other programs.
- The number of ticks that DOS runs, keeping the Print program from working. You can control this with the /S parameter of the Print command. If you don't specify otherwise, DOS assigns eight timer ticks to /S. Increasing the value of /S gives more time to DOS and less time to the Print program.
- The number of ticks that the Print program runs, up to the maximum number available (as set by /S), keeping DOS from working. You can control this with the /M parameter of the Print command. If you don't specify otherwise, DOS assigns two timer ticks to /M. Increasing the value of /M gives more time to the Print program and less time to DOS.
- The number of ticks that the Print program waits for a printer that is busy. You can control this with the /U parameter of the Print command. If you don't specify otherwise, DOS assigns one timer tick to /U.

- The size of the printer buffer. Because this is part of the printer (hardware, not software), the only control you have is over which printer you buy (some printer buffers are larger than others) or whether you add a larger printer buffer (some printers offer this as an option).

No matter how fast your printer or computer is, or how accurately you specify the Print command parameters, the printer probably won't print at top speed while you're using the computer to do something else. If it did, DOS would seem frozen most of the time, defeating the main purpose of the Print program. On the other hand, if your computer appeared to run at top speed while you were printing a file, the printer might print one line every 20 or 30 seconds; that, too, would be unacceptable.

The object is to allocate enough memory to the print buffer so that the Print program doesn't have to use the disks too often, and to find values for the /S, /M, and /U parameters of the Print command that split up the ticks between DOS and the Print program so that both system performance and printing speed are acceptable.

Because so many factors affect performance, you might want to experiment: Type the Print command with values for /S, /M, and /U, then print some files and do something else (such as edit a file or work with a spreadsheet) to see how your system performs. You can specify these parameters only the first time you enter the Print command, so if either DOS or the printer are unacceptably slow, you'll have to restart DOS and type the Print command with a different set of values.

Figure 12-5 suggests some values for the /B parameter (size of the print buffer). If you choose one of these values, type the exact number shown; they're all multiples of 512 (the size of a disk sector), which lets DOS use memory as efficiently as possible.

Typical File Size	***Files Printed at One Time***		
	Fewer than 5	***5 to 10***	***More than 10***
Fewer than 4000 bytes	2048	4096	8192
4000 to 8000 bytes	4096	8192	12,288
More than 8000 bytes	8192	12,288	16,384

Figure 12-5. *Suggested values for the /B parameter of the Print command.*

If you want to experiment, choose a value for /B from Figure 12-5 and use the following starting values for /S, /M, and /U:

```
/S:25 /M:5 /U:2
```

If your printer's buffer is larger than 2K (this is hardware, remember; the printer manual should tell you the size), start with a larger value for /M to give the Print program time enough to fill the buffer. The ratio of /S to /M determines how DOS and the Print program split the available time, so increase /S to keep the same starting proportions:

```
/S:35 /M:7 /U:2
```

In both of these sets of suggested starting values, the Print program gets 20 percent of the available time. To speed up DOS, increase /S or decrease /M or /U. To speed up the printer, decrease /S or increase /M or /U. The higher the /S and /M values, the less time is lost switching between DOS and the Print program. But, with higher /S and /M values, the jerkier DOS and the Print program may work. Some changes in the parameters won't seem to have much effect, and the printer may pause at odd moments almost independently of the settings, so it probably isn't worth spending a lot of time trying to find the exact settings that give the best performance. But with just a few tries you can probably improve the performance over the values that DOS assumes.

Printing a File in Pages

Sending a long file to a dot-matrix printer with the Copy command prints continuously, right over the perforations between pages. This is pretty ugly, so word processors and other application programs skip over the perforations (as does the Print command). If you'd rather not fire up your word processor or use the Print program to print a file, you can use a batch file that prints a file while skipping the perforations.

Laser printers don't have the same problem. Since a laser printer is, by definition, a sheet-feed printer, the text is naturally formatted to a *logical page* area within the physical page. This logical page area, however, can be adjusted just as a dot-matrix printer's area can be changed in size.

Most printers recognize commands that specify the length of the page and how many lines to skip at the bottom to avoid the perforation. PRNT.BAT is a batch file that prints a file in pages by sending these commands to the printer and then copying the file to the printer. Figure 12-6 explains the Epson dot-matrix printer commands used in PRNT.BAT.

Figure 12-7 displays Hewlett-Packard print commands, using the PCL printing language. These commands are comparable to those listed in Figure 12-6 for Epson printers, but notice that there are no Tab Set commands. Instead, the PCL uses complex cursor-positioning commands, such as the Horizontal Cursor Positioning command shown in Figure 12-7.

Printer Commands	*Description*
Form Feed	*character 12* Advances the paper to the top of the next page.
Set Left Margin	ESC l *col* Sets the left margin to the specified column.
Set Page Length	ESC C *0 inches* Sets the page length to the specified number of inches.
Skip Perforation	ESC N *lines* Skips the specified number of lines at the bottom of each page. If you adjust the paper in the printer to the top of a page, the perforation skip leaves top and bottom margins equal to half the perforation skip. The perforation skip is always calculated in lines, regardless of the line spacing, so the perforation skip gets pretty small if you specify a perforation skip of a few lines plus narrow line spacing.
Tab Set	ESC D *tab1 tab2 . . . character 0* Sets the tabs to the columns specified by *tab1*, *tab2*, and so on up to 32 tabs in a line. Specify each tab position with the character whose code corresponds to the column number: column 1 would be Alt-1 and column 10 would be Alt-10. End the list of tab positions with *character 0*.
Tab Set (repeating)	ESC e *character 0 cols* Sets a tab position every specified number of columns across the line. ESC e *character 0 character 5* for example, sets a tab at columns 5, 10, 15, and so on. Most printers accept Alt-128 as well as *character 0*. Some printers call this Set Tab Unit.

Figure 12-6. *Epson printer commands used to print in pages.*

Printer Commands	*Description*
Form Feed	*character 12* Advances the paper to the top of the next page
Set Left Margin	ESC &a*col*L Sets the left margin to the specified column
Top Margin	ESC &l*lines*E Sets the top margin to the specified line
Page Size	ESC &l*val*A Sets the size of the page: *2* = letter, *3* = legal
Set Text Length	ESC &l*lines*F Sets the size of the text area on the page
Set Page Length	ESC &l*lines*P Sets the page length to the specified number of lines
Horizontal Cursor Positioning	ESC &a*col*C Sets the cursor to column *col*, as measured from the first column, which is 1

Figure 12-7. *PCL printer commands used to print in pages.*

That command moves the cursor to a specific column on the screen—not a tab stop.

PRNT.BAT also prints the name of the file and the current time and date at the top of the first page, and accepts wildcard characters. It's an easy way to handle quick print jobs such as rough drafts of letters or several batch files.

Most printers also accept Alt-128 where *character 0* is shown in Figure 12-6. Because Alt-128 can be entered using the Alt-key technique, it is used in place of *character 0* in this book. If your printer doesn't accept Alt-128, substitute *character 0.* (For information on using the Alt-key technique to type a non-keyboard character, see Chapter 2.)

Remember, these commands are for Epson-compatible and HP-compatible printers. If your printer uses different commands, these won't have the intended effect. Your printer should have corresponding commands, however; check the manual and substitute the commands it recognizes.

Figure 12-8 shows the commands in PRNT.BAT. The version shown uses Epson printer commands; HP PCL commands are listed after the figure.

If you have a Hewlett-Packard or compatible laser printer, change lines 18, 21, and 23 in the file to read:

```
18:     echo E(10UE&l2AE&l66FE&a1L > prn
21:     echo ||FILE: %1E&a24C||  > prn
23:     echo E&a5L > prn
```

PRNT.BAT uses the technique described under "A Batch File Can Call Itself" in Chapter 6 to carry out several commands for each of a series of files. It does this by using the Call command to call itself for each file, specifying the batch-file name as the first parameter and *print* as the second parameter.

Create PRNT.BAT with your text editor or word processor and type the commands shown in Figure 12-8. If you're using an HP-compatible printer, remember to replace lines 18, 21, and 23 with their appropriate counterparts. The following explanations describe how PRNT.BAT works:

- Line 2 skips the following error message if you specify the name of a file as the parameter with the Print command.
- Lines 3 through 5 warn you that you must specify the name of a file, and then return to DOS.

```
 1:    @echo off
 2:    if not "%1"=="" goto OK
 3:    echo [7mYou must specify a file name.[m
 4:    echo Please re-enter the command.
 5:    goto END
 6: :OK
 7:    if "%2"=="print" goto PRINT_IT
 8:    date < c:\batch\cr.dat : find "C" > stamp.$$$
 9:    time < c:\batch\cr.dat : find "C" >> stamp.$$$
10:    for %%p in (%1) do call prnt %%p print
11:    erase stamp.$$$
12:    goto END
13: :PRINT_IT
14:    if exist %1 goto GOT_FILE
15:    echo [7mI can't find a file named %1[m.
16:    goto END
17: GOT_FILE
18:    echo [C<Alt-128><Alt-11>[N<Alt-6>[1<Alt-1>[D<Alt-23>
       <Alt-128>> prn
19:    copy stamp.$$$ prn > nul
20:    echo ╔═══════════════════════╗ > prn
21:    echo ‖FILE: %1 <tab> ‖  > prn
22:    echo ╚═══════════════════════╝ > prn
23:    echo [1<Alt-5>[e<Alt-128><Alt-8>> prn
24:    copy %1 prn > nul
25:    echo <Alt-12>> prn
26:    echo ** File [7m%1[m printed
27: :END
```

Figure 12-8. *PRNT.BAT: Print files with title and date stamp.*

- Line 7 skips directly to the commands that print the file if the second parameter is *print*. The commands in lines 8 through 12 are carried out only the first time PRNT.BAT runs; they are skipped each time PRNT.BAT calls itself to print a file.
- Lines 8 and 9 create a temporary file named STAMP.$$$ that contains a date and time stamp. The file CR.DAT is a text file that contains a single carriage return. (See Chapter 6 for details of creating this file.)
- Line 10 uses the Call command to call PRNT.BAT once for each file specified by %1. Each time PRNT.BAT calls itself, it specifies *print* as the second parameter so that the If command in line 7 will skip the commands in lines 8 through 12 and just print the file.
- Line 11 erases the temporary file that contains the date and time stamps because they're no longer needed.

- Line 12 returns to DOS after all the files have been printed. This is the actual end of the batch file.
- Line 14 skips the following error message if the file you specified exists.
- Lines 15 and 16 warn you that the file you specified doesn't exist, and then return to DOS.
- Line 18 sends four commands to the printer:

Command	*Description*
⎋*C<Alt-128><Alt-11>*	Sets the page length to 11 inches
⎋*N<Alt-6>*	Sets the perforation skip to six lines
⎋*l<Alt-1>*	Sets the left margin to column 1
⎋*D<Alt-23><Alt-128>*	Sets one tab at column 23

For HP printers, the following four commands are sent:

Command	*Description*
⎋*(10U*	Uses the PC-8 symbol set (with the extended ASCII characters)
⎋*&l2A*	Sets the paper size to *2*, letter (8½ by 11 inches)
⎋*&l66F*	Sets the text length to 66 lines
⎋*&a1L*	Sets the left margin to column 1

The second and third commands, Page Size and Text Length, aren't really necessary; they're usually the default settings for the printer.

- Line 19 prints the date and time stamps.
- Lines 20 through 22 print a title block at the top of the page that contains the name of the file. The block uses the following box-drawing characters:

╒ Alt-201	╚ Alt-200	= Alt-205
‖ Alt-186	╕ Alt-187	╝ Alt-188

  Lines 20 and 22 both contain 20 <*Alt-205*> characters.

  Press the Tab key once between *%1* and <*Alt-186*> in line 21. Line 18 sets one tab at column 23 so that the right edge of the title block is in the correct position regardless of the length of the file name.

  For HP-compatible printers, the tab stops are all at eight-column intervals. To position character <Alt-186> properly, the Horizontal Cursor Position command is used. That command sets the cursor to column 24, where character <Alt-186> will be properly positioned.

- Line 23 sends two commands to the printer:

Command	*Description*
E*l<Alt-5>*	Sets the left margin to column 5
E*e<Alt-128><Alt-8>*	Sets a tab every eight columns across the line

For HP printers, the following command is sent:

Command	*Description*
E*&a5L*	Sets the left margin to column 5

- Line 24 copies the file to the printer.
- Line 25 sends a Form Feed command to the printer, advancing the paper to the top of the next page or ejecting a page from a laser printer. When you enter *<Alt-12>*, your word processor may break the line and display a new page mark. This looks like trouble, but don't worry: The correct characters are in the file. Some WordStar-like text editors, such as Edit, display the Ankh character, ♀.
- Line 26 displays the name of the file that was printed.
- Line 27 ends the copy of PRNT.BAT that was called to print a file. It returns to the Call command in line 10. If there is another file to be printed, another Call command is carried out, which calls PRNT.BAT again; when there are no more files to be printed, the Goto command in line 12 ends PRNT.BAT.

Testing PRNT.BAT

Test PRNT.BAT with a short text file; if there isn't one in the current directory, create it by copying text from the console. The examples assume you have a file named TEST.DOC. First, make sure that the printer is turned on and the paper is adjusted to the top of a page. Now, type the PRNT.BAT command with no parameter:

```
C>prnt
```

PRNT.BAT should warn you to specify a file name and return to DOS. Type the command again, and this time specify TEST.DOC:

```
C>prnt test.doc
```

After some disk activity, the printer should print the current date and time, the title block with the name of the file, and then the file itself.

Finally, type the command and specify a file that doesn't exist:

```
C>prnt 123.456
```

PRNT.BAT should tell you that it can't find that file, and then return to DOS.

You can use PRNT.BAT to print a series of files by using a wildcard character in the file name. To print all the files with the extension DOC, for example, you would type *prnt *.doc*.

If you prefer a particular combination of typeface, line spacing, or other formatting features, put the commands in line 23 of PRNT.BAT. You could even create separate versions of PRNT.BAT for printing with different combinations of commands.

You can incorporate the Printer Setup menu, described later in this chapter, into PRNT.BAT by adding a few commands to the batch file. The Printer Setup menu lets you control the type of print, line spacing, and other printer features each time you print a file. See "Combining the Printer Setup Menu and PRNT.BAT" at the end of this chapter.

Printing Really Small

When the occasion demands, your printer can print quite small yet remarkably readable characters packed closely together. Figure 12-9 shows an example of this output.

Figure 12-9 is printed in compressed superscript on an Epson-compatible printer. The characters are the same width as compressed print (17 per inch) but not as high, so the line spacing is reduced. Standard line spacing is six lines per inch, or 36/216 inch; the line spacing in Figure 12-7 is 15/216 inch. This combination yields a maximum of 133 characters in a line and 14.4 lines per inch, or 144 lines on an 11-inch page with ½-inch top and bottom margins.

Printing this small requires changing both the type of print and the line spacing. Two new Epson printer commands are used:

Printer Command	***Description***
Set Line Spacing (in 1/216 inch)	[E]3 *units* Sets the line spacing to the specified number of 1/216 inch.
Start Superscript	[E]S0 Starts printing superscript print, a smaller version of the typeface currently in effect that is printed higher on the line than normal. If you change the typeface without stopping superscript, the printer prints the superscript form of the new typeface.

Similar commands are used on Hewlett-Packard and compatible laser printers:

Printer Command	***Description***
Vertical Motion Index	⎋&l*units*C Sets the line spacing to the specified number of 1/48 inch. (See also the Line Spacing command in Appendix B.)
Pitch (characters per inch)	⎋(s*pitch*H Sets the number of characters per inch, or pitch.

STANDARD SCREW THREADS

The Unified and American Screw Threads included in Table 87 are taken from the publication of the American Standards Association, ASA B1.1--1949. The coarse-thread series is the former United States Standard Series. It is recommended for general use in engineering work where conditions do not require the use of a fine thread. The fine-thread series is the former "Regular Screw Thread Series" established by the Society of Automotive Engineers. The fine-thread series is recommended for general use in automotive and aircraft work, and where special conditions require a fine thread. The extra-fine-thread series is the same as the former SAE fine series and the present SAE extra-fine series. It is used particularly in aircraft and aeronautical equipment where (1) thin-walled material is to be threaded; (2) thread depth of nuts clearing ferrules, coupling flanges, etc., must be held to a minimum; and (3) a maximum practicable number of threads is required within a given thread length.

TABLE 87. STANDARD SCREW THREADS

Sizes	Basic Major Diameter D	Thds per Inch N	Basic Pitch Diameter * E	Minor Diameter Ext Thds K_s	Minor Diameter Int Thds K_n	Section at Minor Diameter at D - $2h_b$	Stress Area §
	Inches		Inches	Inches	Inches	Sq In.	Sq In.
Coarse-thread Series--UNC and NC (Basic Dimensions)							
1(.073)	0.0730	64	0.0629	0.0538	0.0561	0.0022	0.0026
2(.086)	0.0860	56	0.0744	0.0641	0.0667	0.0031	0.0036
3(.099)	0.0990	48	0.0855	0.0734	0.0764	0.0041	0.0048
4(.112)	0.1120	40	0.0958	0.0813	0.0849	0.0050	0.0060
5(.125)	0.1250	40	0.1088	0.0943	0.0979	0.0067	0.0079
6(.138)	0.1380	32	0.1177	0.0997	0.1042	0.0075	0.0090
8(.164)	0.1640	32	0.1437	0.1257	0.1302	0.0120	0.0139
10(.190)	0.1900	24	0.1629	0.1389	0.1449	0.0145	0.0174
12(.216)	0.2160	24	0.1889	0.1649	0.1709	0.0206	0.0240

* British: Effective Diameter
§ The stress area is the assumed area of an externally threaded part which is used for the purpose of computing the tensile strength.

Figure 12-9. *How small is small?*

You need only three printer commands to print this small on an Epson printer:

Command	*Description*
<*Alt-15*>	Starts compressed print
ESO	Starts superscript print
E3<*Alt-15*>	Sets line spacing to 15/216 inch

Two commands are used on the HP laser printer:

Command	*Description*
E(s16.66H	Sets pitch to 16.66 characters per inch (small)
E&l4C	Sets line spacing to 4/48 inch

It takes only a minute to try these commands with your printer. First, create a batch file named TINY.BAT or TINYPCL.BAT and type the two lines shown in Figure 12-10 for Epson printers, or Figure 12-11 for HP-compatible printers.

```
1:  @echo off
2:  echo <Alt-15>ESOE3<Alt-15> > prn
```

Figure 12-10. *TINY.BAT: Printing really small on an Epson printer.*

```
1:  @echo off
2:  echo E(s16.66HE&l4C > prn
```

Figure 12-11. *TINYPCL.BAT: Printing really small on a Hewlett-Packard laser printer.*

If you don't have a short text file in the current directory, copy from the console to create one named SHORT.DOC:

```
C>copy con short.doc
This is a file to see
just how small the printer
can print.  How small is it?
^Z
        1 file(s) copied
```

If you have an Epson-compatible printer, type the TINY.BAT command and copy the test file to the printer. (If you're using a file of your own, substitute its name for SHORT.DOC.)

```
C>tiny
C>copy short.doc prn
        1 file(s) copied
```

If you have a Hewlett-Packard or compatible laser printer, use the TINY-PCL.BAT program with SHORT.DOC (or your favorite text file) and enter the following:

```
C>tinypcl
C>copy short.doc prn
        1 file(s) copied

C>echo ^L > prn
```

That final Echo command is used to echo a Ctrl-L character to the printer, which ejects the page so you can see it. (That's character <Alt-12>, not a caret and L character.)

The printer should print the file. You might have to look closely.

> ❑ **NOTE:** *Some word processors let you specify the font (type of print) and line spacing. Figure 12-9, for example, was printed using Microsoft Word, specifying a font of six-point Pica Super-script and a line spacing of five points (one point equals 1/72 inch or 3/216 inch, so five points is 15/216 inch).*

You probably wouldn't use such small printing every day, but it might be just the thing to print that huge spreadsheet or to make a permanent record of some data you don't refer to very often, or even to print your New Year's resolutions.

Sending Commands to the Printer with a Program

In addition to redirecting the output of the Echo command and copying a file to the printer, you can use a program to send commands to the printer. Such programs can be quite small, often less than 20 bytes, and have the advantage of not sending a linefeed as the Echo command does.

But a program is stored in a file, and any file, no matter how small, takes up the minimum amount of space required by the type of disk on which it's stored. A 1-byte file, for example, usually takes up 2048 (2K) bytes on a hard disk. You just can't win.

So even if you limited yourself to 25 of the most common printer commands and combinations, the printer command files would take up 25K

bytes on a 360K floppy disk, or 50K to 100K bytes on a hard disk. That's a lot of disk space to give up just to avoid adjusting the paper in the printer to the top of a page.

That's why, with one exception, this book doesn't use programs to send commands to the printer. The exception is due to an idiosyncrasy of the IBM Graphics Printer (originally the Epson MX-80): Unlike almost all other Epson and Epson-compatible printers, you can't reset the IBM Graphics Printer with the ESC@ printer command.

A Program to Reset the IBM Graphics Printer

PRNTREST.COM is a small assembly-language program that resets all Epson-compatible printers, including the IBM Graphics Printer. If your printer resets when you send it ESC@, you can skip this section and go on to "The Printer Setup Menu."

You create PRNTREST.COM using the Debug command. The procedure is described in Chapter 5. If you haven't yet gone through the examples in Chapter 5, you might do so before continuing here.

Using your text editor, word processor, Edit, or Edlin, create a file named PRNTREST.SCR and enter the lines shown in Figure 12-12:

```
1:  a 100
2:  mov  ah,1
3:  sub  dx,dx
4:  int  17
5:  int  20
6:
7:  n prntrest.com
8:  r cx
9:  8
10:  w
11:  q
```

Figure 12-12. *PRNTREST.SCR: Resetting the IBM Graphics Printer.*

Proofread PRNTREST.SCR against Figure 12-12. Make certain that line 6 is blank and that the last line is a *q*; if there is an error in either of those lines, you'll have to restart DOS when you try to create PRNTREST.COM. When you're sure there are no differences, save PRNTREST.SCR.

To create PRNTREST.COM, type the following Debug command:

```
C>debug < prntrest.scr
```

If nothing seems to happen and pressing a key has no effect, restart DOS, proofread PRNTREST.SCR against Figure 12-12, correct any differences, and retype the Debug command.

DOS should respond as shown in Figure 12-13. To check this, proofread the display against Figure 12-13. If you find any differences, proofread PRNTREST.SCR against Figure 12-12, correct any differences, and retype the Debug command.

```
-a 100
xxxx:0100  mov ah,1
xxxx:0102  sub dx,dx
xxxx:0104  int 17
xxxx:0106  int 20
xxxx:0108
-n prntrest.com
-r cx
CX 0000
:8
-w
Writing 0008 bytes
-q
```

Figure 12-13. *DOS's response when creating PRNTREST.COM.*

If the screen agrees with Figure 12-13, try PRNTREST.COM. Make sure the printer is on, and type the PRNTREST command:

```
C>prntrest
```

You should hear the sound the printer makes when you turn the system on or restart DOS.

The Printer Setup Menu

The remainder of this chapter describes the most ambitious batch file yet in this book—one that makes your printer more useful by letting you send commands to it by pressing only one key. It's also a good demonstration of batch-file techniques, and it creates a major part of the menu system described in Chapter 15.

The Printer Setup menu is actually two menus in one. It lets you switch between two sets of options: one set that selects the typeface, and another set that controls formatting characteristics such as line spacing and page length. A separate window gives instructions for using the menu.

Figures 12-14 and 12-15 show the Typeface commands and Page Format commands that are displayed on the Printer Setup menu.

As Figures 12-14 and 12-15 show, the basic form of the menu remains unchanged; only the choices for function keys F1 through F8 and the title in the instruction window change.

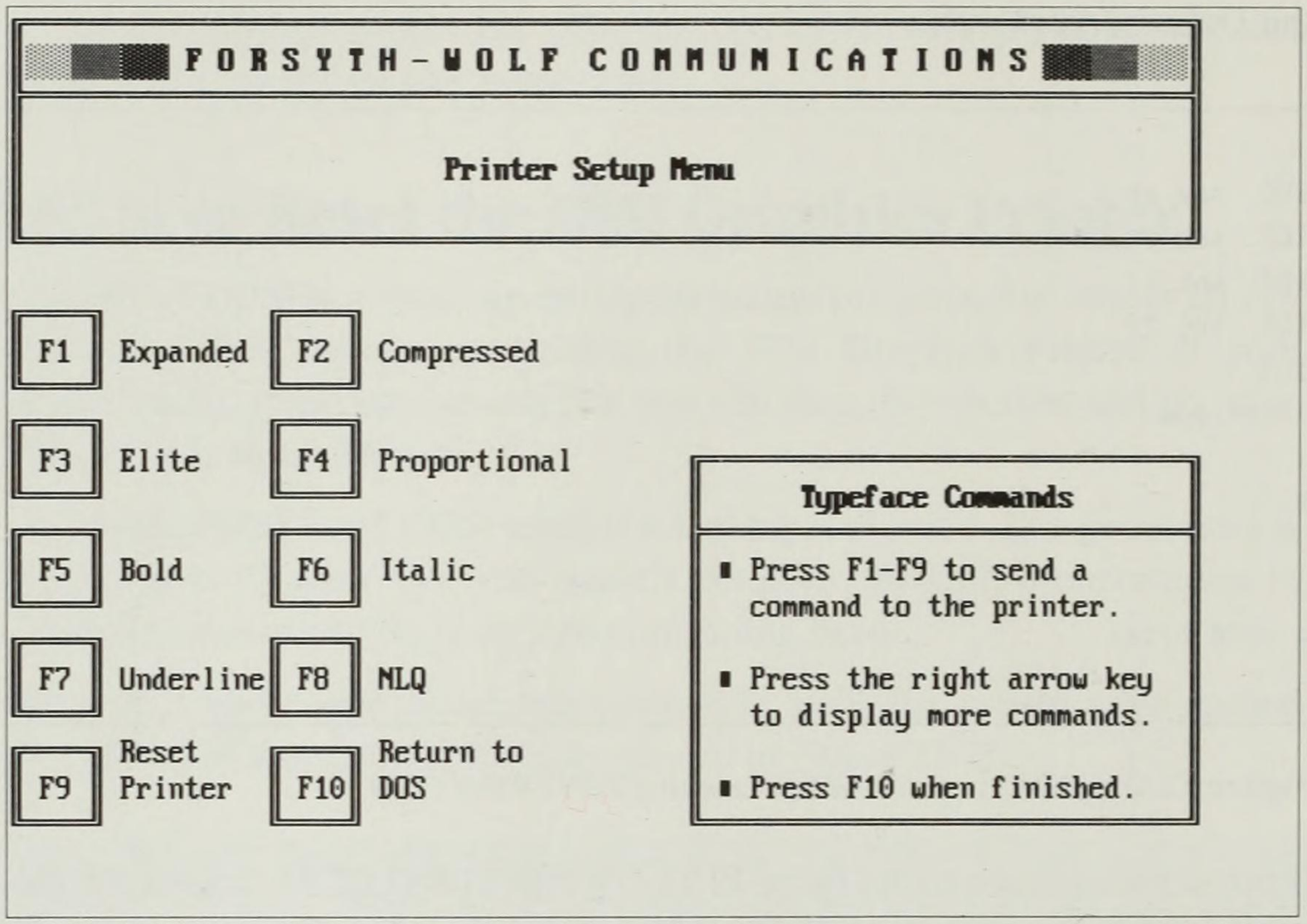


Figure 12-14. *Typeface commands on the Printer Setup menu.*

The Printer Setup menu is simple to operate: You press the function keys to send commands to the printer, press the right-arrow key to display the other set of commands, and press F10 when you're ready to print. All that remains is to position the paper at the top of the page and copy the file to the printer.

You can simplify this even further by calling the Printer Setup menu from PRNT.BAT; see "Printing a File in Pages" earlier in this chapter and "Combining the Printer Setup Menu and PRNT.BAT" at the end of this chapter.

The Printer Setup menu requires one batch file and three text files:

- PRNTMENU.BAT contains the commands that operate the menu.
- PRNTMENU.DOC contains the unchanging part of the menu.

- PRNTOPT1.DOC contains the Typeface commands options.
- PRNTOPT2.DOC contains the Page Format commands options.

It also requires REPLY.COM. If you haven't yet created this program, see "Making Batch Commands Interactive: REPLY.COM" in Chapter 7.

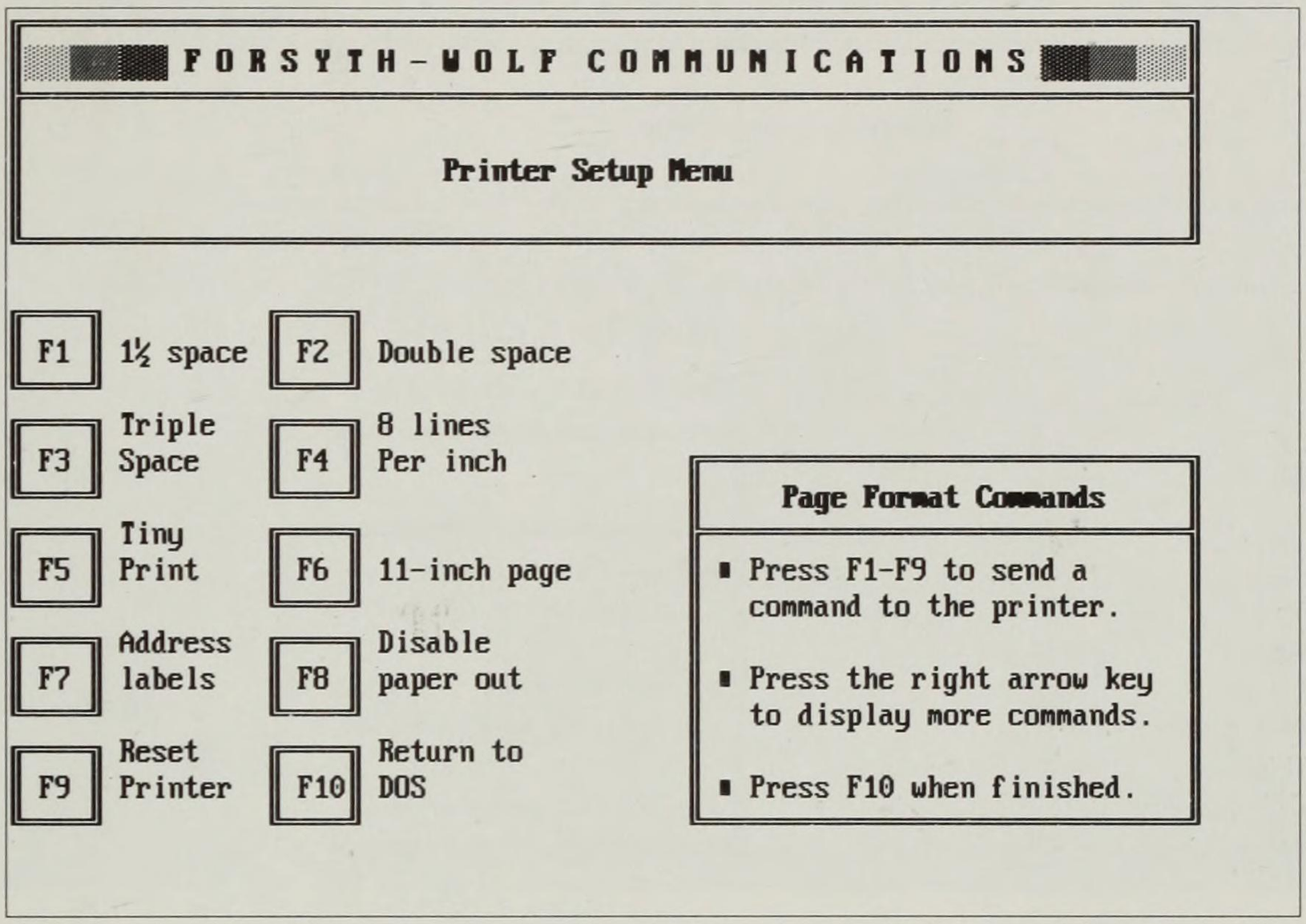


Figure 12-15. *Page Format commands on the Printer Setup menu.*

Entering the files

Although the batch file and text files may look forbidding at first, they're not difficult to enter. The description for each file includes the Alt-key codes for the box-drawing characters and suggests ways that you can use the cut-and-paste capability of your word processor or text editor to simplify the job.

After you enter each text file, you can make sure it's correct by displaying it with the DOS Type command. If, partway through, you're not sure you have typed something correctly, just save the file, exit to DOS, display the file, and compare it to the figure that illustrates it.

It will go much more quickly than you might think.

Entering PRNTMENU.DOC

Figure 12-16 shows the contents of PRNTMENU.DOC. The line numbers, as usual, are for reference only.

```
1:
2:  E[1m ▒▓▓ F O R S Y T H - W O L F  C O M M U N I C A T I O N S ▓▓▒ E[m||
3:
4:
5:                    E[1mPrinter Setup Menu                        E[m||
6:
7:
8:
9:
10:  F1                    F2
11:
12:
13:  F3                    F4
14:
15:
16:  F5                    F6
17:
18:
19:  F7                    F8
20:
21:  Reset                 Return to
22:  F9  Printer           F10 DOS
23:
```

```
▶ Press F1-F9 to send a
  command to the printer.

▶ Press the right arrow key
  to display more commands.

▶ Press F10 when finished.
```

Figure 12-16. *PRNTMENU.DOC: The Printer Setup menu without options.*

If your word processor has line-drawing capabilities, this should be a cinch. Otherwise, use the cut-and-paste capability of your word processor to make the job easier:

1. Type line 1, using the following Alt-key codes for the box-drawing characters:

   ╔ Alt-201 ═ Alt-205 ╗ Alt-187

   <Alt-205> is used 72 times in this line. If your word processor or editor lets you copy characters within a line, type <Alt-205> six times and copy it 11 times.

2. Type line 2, using the following Alt-key codes for the box-drawing and shading characters:

   ‖ Alt-186 ▒ Alt-176 ▓ Alt-177 ▓ Alt-178

   Although line 2 is shown on two lines, enter it as one line; there is one space between the *S* at the end of *COMMUNICATIONS* and the first shading character. You should change the title, of course, to herald your own enterprise. The ANSI.SYS commands display the title in high

intensity, then return to normal. Because ANSI.SYS commands are so long, the final <Alt-186> is seven spaces past the end of line 1.

3. Copy line 1 to line 3. Change the first character to <Alt-204> and the last character to <Alt-185>. The Alt-key codes for the box-drawing characters are

   ╠ Alt-204 ╣ Alt-185

4. Type line 4.

5. Type line 5. Although it is shown on two lines, type it as one line; there are 29 spaces between the *u* at the end of *Menu* and the following ╣. The ANSI.SYS commands display the subtitle in high intensity, then return to normal. Because ANSI.SYS commands are so long, the final <Alt-186> is seven spaces past the end of line 3.

6. Copy line 4 to line 6.

7. Copy line 1 to line 7. Change the first character to <Alt-200> and the last character to <Alt-188>. The Alt-key codes for the box-drawing characters are

   ╚ Alt-200 ╝ Alt-188

8. Line 8 is blank.

9. Type lines 9 through 11. The Alt-key codes for the box-drawing characters are shown in steps 1 and 7.

10. Copy lines 9 through 11 four times. Now you have all 10 function keys.

11. Correct the numbers inside the function-key boxes in lines 13, 16, 19, and 22.

12. Add the first line of the instruction window to line 13. There are 20 spaces between the edge of the F4 box and the upper left corner of the instruction window, and 30 <Alt-205> characters across the top of the instruction window. Align the window frame with this top border.

13. Add the second line of the instruction window to line 14. Copy from the lower right corner of the F4 symbol through the end of the line to the end of lines 15 through 22, then use the overstrike feature of your word processor to change the lines to match Figure 12-16. The Alt-key codes for the box-drawing characters are shown in steps 1, 2, and 3. The ► character is <Alt-16>.

14. Copy line 13 from the edge of the F4 symbol through the end of the line to the end of line 23. Change the lower-left-corner character to <Alt-200> and the lower-right-corner character to <Alt-188>.

15. Add the choices for F9 and F10 in lines 22 and 23.

Save PRNTMENU.DOC, exit to DOS, display the file with the Type command, and compare the display to Figure 12-14 (ignore the menu choices and instruction-window title). Correct any differences, and go on to enter the option files.

Entering PRNTOPT1.DOC and PRNTOPT2.DOC

PRNTOPT1.DOC and PRNTOPT2.DOC display the choices for F1 through F8. They include ANSI.SYS commands to position the choices. Figure 12-17 shows the contents of PRNTOPT1.DOC and PRNTOPT2DOC.

PRNTOPT1.DOC

```
1:   [14;49H[1m    Typeface Commands    [m
2:   [9;8H          [9;24H
3:   [10;8HExpanded [10;24HCompressed
4:   [12;8H         [12;24H
5:   [13;8HElite    [13;24HProportional
6:   [15;8H         [15;24H
7:   [16;8HBold     [16;24HItalic
8:   [18;8H         [18;24H
9:   [19;8HUnderline[19;24HNLQ
```

PRNTOPT2.DOC

```
1:   [14;49H[1m      Page Format Commands [m
2:   [9;8H           [9;24H
3:   [10;8H11/2 space [10;24HDouble space
4:   [12;8HTriple     [12;24H8 lines
5:   [13;8HSpace      [13;24HPer inch
6:   [15;8HTiny       [15;24H
7:   [16;8HPrint      [16;24H11-inch page
8:   [18;8HAddress    [18;24HDisable
9:   [19;8HLabels     [19;24Hpaper out
```

Figure 12-17. *PRNTMENU option files (PRNTOPT1.DOC and PRNTOPT2.DOC).*

As you can see, these two files consist primarily of ANSI.SYS commands and spaces. The spaces are necessary because the choices for corresponding function keys in the two different command sets aren't the same length; if the spaces weren't included, the extra characters in the longer choices would remain on the screen when the shorter choice was displayed.

Because the two files are so similar, it's pretty easy to enter them. Start by creating PRNTOPT1.DOC.

1. Type line 1. The ANSI.SYS command displays the instruction-window title in high intensity. There are three spaces between [E][1m and *Typeface Commands*. There are four spaces between the *s* of *Commands* and the following [E].
2. Type line 2. There are nine spaces between the *H* at the end of [E][9;8H and the following [E]. Although they aren't shown, there are 15 spaces at the end of the line; be sure to enter them.
3. Copy line 2 seven times.
4. Using the overstrike feature of your word processor, change lines 3 through 9 to match PRNTOPT1.DOC in Figure 12-17. Be sure not to change the number of spaces in the middle or at the end of any line.
5. Save PRNTOPT1.DOC, exit to DOS, and display it with the DOS Type command. You should see two columns of choices and *Typeface Commands* to the right of the choice *Bold*.
6. Copy PRNTOPT1.DOC to create PRNTOPT2.DOC.
7. Edit PRNTOPT2.DOC and, using overstrike, change it to match PRNTOPT2.DOC in Figure 12-17. Only the wording of the choices and the instruction-window title change; the ANSI.SYS commands are the same. Again, be sure not to change the number of spaces in the middle or at the end of any line.
8. Save PRNTOPT2.DOC, exit to DOS, and display it with the DOS Type command. You should see two rows of choices and *Page Format Commands* to the right of the word *Print*. (You may have to clear the screen first.)

Now on to the batch file.

The Printer Setup Menu Batch File

PRNTMENU.BAT displays the Printer Setup menu, sends commands to the printer when you press a function key, changes the menu options if you press the right-arrow key, and returns to DOS when you press F10. Except for the number of choices, it's basically an extension of the batch file MENU-6.BAT described in Chapter 7 and shown in Figure 7-6.

Instead of using Echo commands, PRNTMENU.BAT uses Type commands in order to display the menu (PRNTMENU.DOC) as well as the options (PRNTOPT1.DOC and PRNTOPT2.DOC). Not only is this faster, it holds down the size of PRNTMENU.BAT. In general, the larger the batch file, the harder it is to remember just how it works or where everything happens.

There's a penalty in disk space for not using Echo commands, but the problem isn't severe: Combining all four files into PRNTMENU.BAT would save you 2K bytes if you use floppy disks and from 4K to 12K bytes if you use a hard drive. The saving in disk space, however, probably isn't worth the increased size of the batch file. If, after entering and using the Printer Setup menu, you want to merge all four files, see "Combining the DOC Files with PRNTMENU.BAT" later in this chapter.

Figure 12-18 shows the commands in PRNTMENU.BAT for Epson-compatible printers, and Figure 12-19 shows the commands you should use for a Hewlett-Packard–compatible laser printer.

```
1:    @echo off
2:    cls
3:    type prntmenu.doc
4:  :CHOICE_1
5:    type prntopt1.doc
6:  :REPLY_1
7:    reply
8:    if errorlevel 77 if not errorlevel 78 goto CHOICE_2
9:    if errorlevel 69 goto REPLY_1
10:   if errorlevel 68 goto END
11:   if errorlevel 67 echo E@> prn
12:   if errorlevel 66 if not errorlevel 67 echo En> prn
13:   if errorlevel 65 if not errorlevel 66 echo E-1> prn
14:   if errorlevel 64 if not errorlevel 65 echo E4> prn
15:   if errorlevel 63 if not errorlevel 64 echo EE> prn
16:   if errorlevel 62 if not errorlevel 63 echo Ep1> prn
17:   if errorlevel 61 if not errorlevel 62 echo EM> prn
18:   if errorlevel 60 if not errorlevel 61 echo <Alt-15>> prn
19:   if errorlevel 59 if not errorlevel 60 echo EW1> prn
20:   goto REPLY_1
21: :CHOICE_2
22:   type prntopt2.doc
23: :REPLY_2
24:   reply
25:   if errorlevel 77 if not errorlevel 78 goto CHOICE_1
26:   if errorlevel 69 goto REPLY_2
27:   if errorlevel 68 goto END
28:   if errorlevel 67 echo E@> prn
29:   if errorlevel 66 if not errorlevel 67 echo E8> prn
30:   if errorlevel 65 if not errorlevel 66 echo EC<Alt-9>> prn
31:   if errorlevel 64 if not errorlevel 65 echo EC<Alt-128><Alt-11>
      EN<Alt-6>> prn
32:   if errorlevel 63 if not errorlevel 64 echo <Alt-15>ESOE3
      <Alt-15>> prn
```

Figure 12-18. *PRNTMENU.BAT: The Printer Setup menu batch file using Epson-compatible printer commands.* *(continued)*

Figure 12-18. *continued*

```
33:     if errorlevel 62 if not errorlevel 63 echo ⌂0> prn
34:     if errorlevel 61 if not errorlevel 62 echo ⌂31> prn
35:     if errorlevel 60 if not errorlevel 61 echo ⌂3H> prn
36:     if errorlevel 59 if not errorlevel 60 echo ⌂36> prn
37:     goto REPLY_2
38:  :END
39:  cls
```

```
 1:     @echo off
 2:     cls
 3:     type prntmenu.doc
 4:  :CHOICE_1
 5:     type prntopt1.doc
 6:  :REPLY_1
 7:     reply
 8:     if errorlevel 77 if not errorlevel 78 goto CHOICE_2
 9:     if errorlevel 69 goto REPLY_1
10:     if errorlevel 68 goto END
11:     if errorlevel 67 echo ⌂E> prn
12:     if errorlevel 66 if not errorlevel 67 echo ⌂(10U> prn
13:     if errorlevel 65 if not errorlevel 66 echo ⌂&d0D> prn
14:     if errorlevel 64 if not errorlevel 65 echo ⌂(s1S> prn
15:     if errorlevel 63 if not errorlevel 64 echo ⌂(s3B> prn
16:     if errorlevel 62 if not errorlevel 63 echo ⌂(s1P> prn
17:     if errorlevel 61 if not errorlevel 62 echo ⌂(s2T> prn
18:     if errorlevel 60 if not errorlevel 61 echo ⌂(s8S> prn
19:     if errorlevel 59 if not errorlevel 60 echo ⌂(s24S> prn
20:     goto REPLY_1
21:  :CHOICE_2
22:     type prntopt2.doc
23:  :REPLY_2
24:     reply
25:     if errorlevel 77 if not errorlevel 78 goto CHOICE_1
26:     if errorlevel 69 goto REPLY_2
27:     if errorlevel 68 goto END
28:     if errorlevel 67 echo ⌂E> prn
29:     if errorlevel 66 if not errorlevel 67 echo ⌂&l00> prn
30:     if errorlevel 65 if not errorlevel 66 echo ⌂&l10> prn
31:     if errorlevel 64 if not errorlevel 65 echo ⌂&l2A> prn
32:     if errorlevel 63 if not errorlevel 64 echo ⌂(s16.66H⌂&l4C> prn
33:     if errorlevel 62 if not errorlevel 63 echo ⌂&l8D> prn
34:     if errorlevel 61 if not errorlevel 62 echo ⌂&l2D> prn
```

Figure 12-19. *PRNTMENU.BAT: The Printer Setup menu batch file using HP-compatible printer commands.* *(continued)*

Figure 12-19. *continued*

```
35:     if errorlevel 60 if not errorlevel 61 echo E&13D> prn
36:     if errorlevel 59 if not errorlevel 60 echo E&14D> prn
37:     goto REPLY_2
38:  :END
39:  cls
```

Entering PRNTMENU.BAT

Most of PRNTMENU.BAT is made up of two nearly identical parts, so just as you used the cut-and-paste capability of your word processor to simplify entering the DOC files, you can simplify typing PRNTMENU.BAT. (Note: Although lines 31 and 32 are shown on two lines in Figure 12-18, type each as one line.) Create PRNTMENU.BAT:

1. Type lines 1 through 8.
2. Copy line 8 eleven times.
3. Change lines 9 through 19 to match Figure 12-18 or 12-19.
4. Type line 20.
5. Copy lines 4 through 20.
6. Change lines 21 through 37 to match Figure 12-18 or 12-19. Only the labels and printer commands change.
7. Type lines 38 and 39. Save PRNTMENU.BAT.

How PRNTMENU.BAT works

As the line-by-line descriptions show, lines 3 through 20 and lines 21 through 37 of PRNTMENU.BAT are the same except for the labels and printer commands:

- Line 3 displays the unchanging part of the menu.
- Line 4 is the first line of the Typeface commands part of the batch file.
- Line 5 displays the options and instruction-window title of the Typeface commands.
- Line 7 waits for you to press a key.
- Line 8 goes to display the Page Format commands if you press the right-arrow key (the key code for the right-arrow key is 77).
- Line 9 goes back to wait for you to press a key if the code of the key you press is from 69 through 77. This is necessary because of the gap in key codes between the check for key code 77 (right arrow) in line 8 and the check for key code 68 (F10) in line 10.

- Line 10 returns to DOS if you press F10.
- Lines 11 through 19 send a command to the printer, depending on which function key you press. For both Epson-compatible and HP-compatible printers, the following commands are set.

Line	*Key Code*	*Function Key*	*Printer Command*
11	67	F9	Reset
12	66	F8	Start Near Letter Quality
13	65	F7	Start Underline
14	64	F6	Start Italic
15	63	F5	Start Bold
16	62	F4	Start Proportional
17	61	F3	Start Elite
18	60	F2	Start Compressed
19	59	F1	Start Expanded

For the HP laser printer, which already prints in Near-Letter-Quality mode or better, line 12 contains the command to activate the PC-8 character set, which contains the IBM extended ASCII characters. (You can make the display reflect this change by editing the file PRNT-OPT1.DOC to display the new command.)

- Line 20 goes back to wait for you to press a key if you pressed any key other than F1 through F10. This is the last command of the Typeface commands part of the batch file.
- Line 21 is the first line of the Page Format commands part of the batch file.
- Line 22 displays the options and instruction-window title of the Typeface commands.
- Line 24 waits for you to press a key.
- Line 25 goes to display the Typeface commands if you press the right-arrow key (the key code for the right-arrow key is 77).
- Line 26 goes back to wait for you to press a key if the code of the key you press is from 69 through 77. This is necessary because of the gap in key codes between the check for key code 77 (right arrow) in line 25 and the check for key code 68 (F10) in line 27.
- Line 27 returns to DOS if you press F10.
- Lines 28 through 36 send a command to the printer, depending on which function key you press. For both Epson-compatible and HP-compatible printers, the following commands are set.

Line	Key Code	Function Key	Printer Command
28	67	F9	Reset
29	66	F8	Disable paper-out detector
30	65	F7	Set page depth to nine lines
31	64	F6	Set page depth to 11 inches, and set perforation skip to six lines
32	63	F5	Start compressed print, start superscript, and set line spacing to 15/216 inch
33	62	F4	Set line spacing to eight lines per inch
34	61	F3	Set line spacing to 108/216 inch (character 108 is l)
35	60	F2	Set line spacing to 72/216 inch (character 72 is H)
36	59	F1	Set line spacing to 54/216 inch (character 54 is 6)

For the HP laser printer, lines 29 and 30 send the codes for portrait and landscape page orientation to the printer (there's no PCL command to disable paper out, and printing address labels on a laser printer is a job better left to professional software). If you find these options are unnecessary, consider changing line 29 to eject a page from the printer by changing the Echo command to *echo <Alt-12>> prn*. You can make the display reflect the changes mentioned here by editing the file PRNTOPT2.DOC.

- Line 37 goes back to wait for you to press a key if you pressed any key other than F1 through F10.

If you print PRNTMENU.BAT in order to proofread it (or just to have a copy of it), be advised: The file contains lots of printer commands, and the printer will carry out each one of them. The printed copy will be, well, *interesting,* but still readable.

Using the Printer Setup Menu

Before you test the Printer Setup menu for the first time, do the following:

- If you created the files on a RAM disk, copy all four to a real disk. Be sure to take the time to do this; you don't want to lose the time you spent entering them.
- Make sure all four files are in the current directory.

- Make sure REPLY.COM is either in the current directory or in a directory in the command path.
- Now start the Printer Setup menu by typing:

```
C>prntmenu
```

DOS should display the Printer Setup menu with the Typeface commands as shown in Figure 12-14. Press the right-arrow key; PRNTMENU.BAT should display the Page Format commands, as shown in Figure 12-15.

Make sure the printer is turned on and the paper is adjusted to the top of a page, or ready to print. Now press F9 to reset the printer. The printer should make the same sound it makes when you turn the computer on or restart DOS; a laser printer may eject a page.

Try some of the other commands. For example, select the Typeface commands (press the right-arrow key if necessary), press F3 for Elite type, press the right-arrow key to select the Page Format commands, press F1 for double-spaced printing, and then press F10 to leave the Printer Setup menu. Now print a short file by copying it to the printer or with PRNT.BAT. The file should be printed double-spaced in the Elite typeface.

If this test worked, you're ready to test all the options. The next two sections describe the options of both the Typeface commands and the Page Format commands of the Printer Setup menu.

The Typeface commands

When the Typeface commands are displayed, as shown in Figure 12-14, F1 through F8 select different typefaces, such as Expanded, Compressed, and Elite. As Figure 12-1 shows, you can combine most of the Typeface commands. To avoid overheating the print head on a dot-matrix printer, however, most printers won't print compressed and bold type at the same time; they cancel compressed and print bold normal type. So if you select both F2 (Compressed) and F5 (Bold), and then print a file, the file will be printed in bold normal-sized type.

F9 resets the printer. Use this menu choice when you start sending commands to the printer or if you change your mind after selecting some options and decide to start over. On HP-compatible laser printers, this command will eject a page from the printer if some data is still left to print.

F10 returns to DOS without resetting the printer, so any choices you make are in effect the next time you print a file. Many application programs, including word processors and spreadsheets, reset the printer each time you

print something, so printer options you select with the Printer Setup menu may have no effect.

The Page Format commands

Figure 12-15 earlier in this chapter shows the Page Format commands. The choices include:

- F1 through F4 set line spacing. The normal line spacing is six lines per inch, the same as a standard typewriter. (In the units of measure that most printers use, six lines per inch is a line spacing of $^{12}/_{72}$ inch, or $^{36}/_{216}$ inch.)

  The Page Format commands offer four options (the first three are more widely spaced than normal, the last is more closely spaced):

Command	*Description*
F1	Space-and-a-half ($^{18}/_{72}$ inch or $^{54}/_{216}$ inch), or four lines per inch
F2	Double space ($^{24}/_{72}$ inch or $^{72}/_{216}$ inch), or three lines per inch
F3	Triple space ($^{36}/_{72}$ inch or $^{108}/_{216}$ inch), or two lines per inch
F4	Eight lines per inch ($^{9}/_{72}$ inch or $^{27}/_{216}$ inch)

- F5 (tiny print) prints very small type with very narrow line spacing. For an example and description of the printer commands involved, see "Printing Really Small" earlier in this chapter.
- F6 (11-inch page) sets the printer page depth to 11 inches and the perforation skip to six lines. If you select F6, return to DOS and copy a file to the printer; the printer won't print over the perforations on continuous-form paper. (If you copy several files to the printer, however, it won't separate the files.) For a description of the printer commands involved and some cautions about printing files this way, see "Printing a File in Pages" and Figure 12-5 earlier in this chapter.
- F7 (Address labels) sets the page depth to nine lines. This prints a file of names and addresses on address labels whose top edges are 1½ inches apart if there is a Form Feed command (or your word processor's New Page command) after each entry in the file. Because address labels come in various sizes, you may have to change the Page Depth command in PRNTMENU.BAT. Measure how far it is from the top of one address label to the top of the next, convert the distance to sixths of an inch, and replace <Alt-9> in line 30 of Figure 12-15 with the result. If the distance is one inch, for example, replace <Alt-9> with <Alt-6>. Remember, this command isn't appropriate for a laser printer.

- F8 (Disable paper out) tells the printer not to stop printing when it's close to the bottom of the page. Use this option if your printer stops printing before the end of page when you print on single sheets, such as letterhead. Note: If you select this option and then print a file longer than one page with a single sheet in the printer, the printer will continue printing beyond the end of the paper and print on the platen.
- F9 (Reset printer) and F10 (Return to DOS) are the same as F9 and F10 when the Typeface commands are displayed.

Changing the Printer Setup Menu

You may want to change some of the options on the Printer Setup menu. You could change F9 and F10, for example, so that they don't have the same effect in both sets of commands, or you might want to replace some of the commands with printer options you use more often.

Changing a menu item

To change a menu item, you must change both the title displayed in either PRNTOPT1.DOC (for the Typeface commands) or PRNTOPT2.DOC (for the Page Format commands) and the command that is carried out in the appropriate If Errorlevel command in PRNTMENU.BAT.

For example, suppose you wanted to call F3 in the Page Format commands *Left Margin 6* instead of *Triple Space*, and you wanted it to set the left margin to column 6 instead of setting the line spacing to triple space. The Epson printer command to set the left margin to column 6 is [Esc]*l*<*Alt-6*>; the HP PCL command is [Esc]&a6L. Make the following three changes:

- Change *Triple* in line 4 of PRNTOPT2.DOC (Figure 12-17) to *Left*, followed by two spaces.
- Change *Space* and the three spaces that follow in line 5 of PRNTOPT2.DOC to *Margin 6*.
- Change *echo* [Esc]*3l*> *prn* at the end of line 34 of PRNTMENU.BAT (Figure 12-18) to *echo* [Esc]*l*<*Alt-6*> *prn*. For a laser printer, change *echo* [Esc]*&l2D*> *prn* at the end of line 34 of PRNTMENU.BAT (Figure 12-19) to *echo* [Esc]*&a6L*> *prn*.

PRNTMENU.BAT carries out only one command per menu choice. You could change it so that a menu selection carried out several commands by substituting a Goto command for the Echo command after the appropriate If Errorlevel command, and then carrying out the required commands at the label named in the Goto command and going back to REPLY1 or REPLY2.

For example, suppose you wanted F8 in the Typeface commands to copy all files with the extensions DOC and TXT from the directory C:\WP to your RAM disk (drive D). You would make the following changes:

1. Change *NLQ* in line 9 of PRNTOPT1.DOC (Figure 12-17) to *Fill D:*.
2. Change [E]*n* in line 12 of PRNTMENU.BAT to *goto COPY*.
3. Insert the following after line 20 of PRNTMENU.BAT (Figure 12-18 or 12-19):

```
:COPY
   copy c:\wp\*.doc d:
   copy c:\wp\*.txt d:
   goto REPLY_1
```

If you change one of the Page Format commands this way, the last command would be *goto REPLY_2* instead of *goto REPLY_1*.

Using programs instead of redirecting the Echo command

If you prefer to use programs rather than redirect Echo commands to send commands to the printer, make sure the COM or EXE files are in the current directory or in a directory that's on the search path, and then replace the Echo command following the appropriate If Errorlevel command with the name of the command file.

For example, to use PRNTREST.COM instead of *echo* [E]*@> prn* to reset the printer, change *echo* [E]*@> prn* to *prntrest* in both lines 11 and 28 of PRNTMENU.BAT (Figure 12-18).

Combining the DOC Files with PRNTMENU.BAT

To incorporate PRNTMENU:DOC into PRNTMENU.BAT, insert *echo* followed by a space at the beginning of each line of PRNTMENU.DOC, and then replace line 3 of PRNTMENU.BAT (*type prntmenu.doc*) with the revised version of PRNTMENU.DOC. To be on the safe side, don't erase PRNTMENU.DOC until you have successfully tested the new version of PRNTMENU.BAT.

To incorporate PRNTOPT1.DOC and PRNTOPT2.DOC into PRNTMENU.BAT, insert *echo* followed by a space at the beginning of each line of both files, and then replace line 5 of PRNTMENU.BAT (*type prntopt1.doc*) with the revised version of PRNTOPT1.DOC and replace line 22 (*type prntopt2.doc*) with the revised version of PRNTOPT2.BAT. Again, play it safe and don't erase PRNTOPT1.DOC or PRNTOPT2.DOC until you have successfully tested the new version of PRNTMENU.BAT.

The Printer Setup menu demonstrates just how much you can do with DOS. It's no little effort to write a batch file like this, but the effort pays off each time PRNTMENU.BAT saves you a bit of time or encourages you to use your printer in a way you might not otherwise try. When you incorporate PRNTMENU.BAT into the menu system described in Chapter 15, you'll have a menu system that manages most of the routine uses of your computer, letting you move from application to application with the press of a key rather than by changing directories and typing commands.

Combining the Printer Setup Menu and PRNT.BAT

If you have entered both PRNT.BAT and the Printer Setup menu, you have two batch files that send commands to the printer and print files by the page. It seems only natural to combine them so that you could use one batch file to print a series of files with the printer commands you want. It's fairly simple; Figure 12-20 shows the commands you insert in PRNT.BAT to incorporate the Printer Setup menu.

```
1:     call prntmenu prnt
2:     echo <Alt-7>Ready to print [1m%1[m. Position paper at top of page.
3:     echo [7mPress F1 to print, any other key to cancel.[m
4:     reply
5:     if errorlevel 59 if not errorlevel 60 goto CONTINUE
6:     echo ** Print canceled.
7:     goto END
8:  :CONTINUE
```

Figure 12-20. *Additions to PRNT.BAT to incorporate the Printer Setup menu.*

Insert the commands in Figure 12-20 after line 7 of PRNT.BAT (Figure 12-8).

- Line 1 uses the Call command to call PRNTMENU.BAT. It specifies *prnt* as a parameter so that PRNTMENU.BAT can determine whether it was run by itself or called by another batch file. As you'll see in a moment, this lets you add a command to PRNTMENU.BAT that changes the wording of the F10 choice.
- Lines 2 and 3 beep and display a message that reminds you to position the paper at the top of a page. The beep is used because if you don't position the paper, the files may not be printed in proper pages. If the beep irritates you, remove the <Alt-7>.

- Line 4 waits for you to press a key.
- Line 5 proceeds to print the files if you press F1.
- Lines 6 and 7 note that you canceled printing and return to DOS.

Save the revised version of PRNT.BAT as PRNT1.BAT and test it by typing *prnt1* followed by the name of a file. You should see the Printer Setup menu, and it should work just as it did when you tested it earlier. Select the printer options you want, then press F10. PRNT.BAT should remind you to position the paper at the top of a page; press F1 to print the file. It's a pretty slick way to print files the way you want them.

One last thing. The option for F10 in the Printer Setup menu reads *Return to DOS*. That isn't true now; when you finish choosing your printer options, PRNTMENU.BAT returns to PRNT.BAT, which prints your file. You can fix that by adding just one command to PRNTMENU.BAT.

The word *Return* is displayed in row 21 of PRNTMENU.DOC and *DOS* is displayed in row 22, so all you have to change is what's displayed in row 22. Because PRNT1.BAT calls PRNTMENU.BAT with the parameter *prnt*, you can use an If command to check for a parameter and, if it's *prnt*, change what's displayed in row 22 with an Echo command.

Insert the following command after line 5 of PRNTMENU.BAT:

```
if "%1"=="prnt" echo [22;24HPrint File
```

Now the F10 choice is accurate, and the chapter is ended. Save the revised version of PRNTMENU.BAT and turn your printer off. It's earned a rest.

CHAPTER

13

Increasing Your Keyboard's IQ

You spend most of your time at the computer with your hands on the keyboard—the primary tool for communicating with DOS and application programs. Any changes that make the keyboard better suited to your needs can pay big dividends. This chapter describes a few such changes; it shows you how to:

- Use the editing keys to correct or reuse the last command
- Work with a command history and macros by using Doskey
- Redefine the keyboard to let you type the box-drawing characters, carry out a DOS command by pressing a single key, or carry out a series of commands by pressing one key
- Speed up the repeat action of an IBM PC/AT, PS/2, or compatible computer

Using the Editing Keys to Edit DOS Commands

DOS doesn't throw away a command after you press the Enter key. It keeps the command in memory in a *template* until you press Enter again. The editing keys, shown in Figure 13-1, behave just as they do in Edlin. You can use them to display the last command you typed and enter it again, either unchanged or with changes.

→ **Display next character**	F2 **Display command up to next occurrence of specified character**
← **Move cursor left one character and erase (also Backspace)**	F3 **Display remainder of command**
Ins **Start inserting characters**	F4 **Delete command up to next occurrence of specified character**
Del **Delete one character**	F5 **Copy current command to buffer**
F1 **Display next character**	

Figure 13-1. *DOS command-line editing keys.*

It takes only a couple of quick examples to show you how the editing keys work. To start, change the directory to \DOS (or to wherever you keep your DOS files).

```
C>cd \dos
```

Start with a too-common occurrence: making a mistake while typing a long command. Type the following Directory command, error and all:

```
C>dor *.com ¦ sort
```

When you press Enter, DOS, of course, chides you for the error:

```
Bad command or file name
```

Oops. You meant *dir*. But don't retype the command, just press the right-arrow key. DOS displays the first character of that last command:

```
C>d
```

The whole command is there, waiting for you to use it again. First correct the error by typing *i*:

```
C>di
```

Now press F3; DOS displays the entire command without the error. Press Enter and DOS carries out the command, just as if you had typed it correctly in the first place.

You have used two of the DOS editing keys, the right-arrow key and F3:

- The right-arrow key displays the next character of the previous command; each time you press it, it displays another character until you reach the end of the command.
- Pressing F3 produces the same result as holding down the right-arrow key until the entire command is displayed: It displays the rest of the command to the end of the line.

These editing keys actually work on what's called a *template*, which is just the last DOS command stored in memory. When you press the right-arrow key or F1, the next character in the template is displayed. Pressing F3 displays the entire template. Any new characters you type overwrite old characters in the template, which is what happened above with the *i* you typed to replace the *o* in the original, bad command.

Now suppose you meant to display the EXE files, not the COM files, and you'd like to sort the directory entries on column 16 (the file size). No problem. Hold down the right-arrow key and let it repeat until the command is displayed up to the period:

```
C>dir *.
```

If you overshoot, just use the left-arrow key or the Backspace key to back up (if you prefer to be cautious, you could press the right-arrow key six times instead of holding it down). Now type *exe* to replace *com*:

```
C>dir *.exe
```

Finally press F3 to display the remainder of the command, press the Spacebar, type */+16*, and press Enter:

```
C>dir *.exe ¦ sort /+16
```

DOS displays the EXE files sorted by size. To display only the COM files, you can do the same thing: press the right-arrow key until the command is displayed up to the period—but wait! A better key to press is F2, the search-for key.

At the start of the line, press F2, then press the E key. DOS displays the command line up to the key you pressed, E:

```
C>dir *.
```

Now type *com* and then F3 to display the rest of the line. Press Enter: DOS displays the COM files sorted by size.

You have used another editing key, F2:

- The F2 key searches the template (previous command) for the next character you press. It works like F3 but displays the previous command only up to the key you pressed.

What if you'd rather sort the files in reverse order? Now you'll have to insert some characters, not replace them. That's why there's a key labeled *Ins*. Press F3, then press the left-arrow key three times; or use the F2 key and then press the plus (+) key. Either way will give you:

```
C>dir *.com ¦ sort /
```

Press the Ins key, then type *r*, a space, and a slash (/):

```
C>dir *.com ¦ sort /r /
```

You have used two more of the editing keys, left arrow and Ins:

- The left-arrow key (or the Backspace key; they're equivalent) moves the cursor one column left and erases the character located there. It doesn't, however, erase the character from memory—you can display the character again by pressing the right-arrow key.

- The Ins key tells DOS to insert the characters you type next, adding them to the command instead of replacing characters in the command.

Press F3 and Enter to complete and carry out the command. Now DOS displays the COM directory entries with the largest file first.

Finally, what if you decide you'd like to see all the files? You'll have to delete *.com* from the command. That's what the key marked Del is for; it's the last editing key. Each time you press Del before displaying the entire command, DOS deletes one character from the command.

Press the right-arrow key four times, or press F2 followed by an asterisk (*):

```
C>dir
```

Now press Del six times. You have to take this operation on faith for a moment, because nothing seems to happen, but press F3 to display the rest of the command:

```
C>dir : sort /r /+16
```

You deleted **.com*. Be careful not to press Enter until you've got the command just the way you want it. Whatever is displayed on the command line when you press Enter becomes the new last command.

Now suppose you want to reverse the reverse sort. To do this, you need to eliminate the */r* switch from the command. Use these steps:

1. Press the right-arrow key 12 times, or press F2 followed by a slash.

2. Press the Del key three times.

Instead of pressing Del three times, you can use the F4 key. F4 works like the F2 key, except instead of displaying a command up to a given character, F4 *deletes* characters. To delete */r* from the command line, first move the cursor up to the slash with the right arrow or F2:

```
C>dir : sort
```

Now press F4 and the slash key again. Nothing will appear to happen, but three characters have just been deleted from the template. Prove it by pressing F3:

```
C>dir : sort /+16
```

The editing keys can save you a lot of keystrokes. After using them for a while, you may never again bang the keyboard in frustration after making

an error in a long command. Of course, the drawback is that you get little visual feedback, which can frustrate long editing attempts. To improve the situation, use the F5 key. F5 copies all your editing into the buffer, allowing you to see your changes and correct any mistakes. Of course, the best approach is to get a command-line editor such as Doskey, which comes with DOS, starting with version 5.

Command-Line Editing with Doskey

If you have DOS 5, you can take advantage of the Doskey program to make command-line editing more versatile and tap a powerful new DOS tool with many capabilities. If you don't have DOS 5, skip ahead to the section "Redefining the Keyboard."

Doskey is a memory-resident keyboard and command-line enhancement utility. Before DOS 5, similar command-line editing programs were available from third parties. Since Doskey is now included with DOS, however, it's the only command-line editor covered here.

Doskey expands on the command-line editing keys already available at the DOS prompt. With Doskey installed, you get the ability to edit command lines in detail, use more convenient keystrokes, and see what you're editing as you edit it. Additionally, Doskey stores old command lines in a *history,* from which they can be recalled for editing or reuse. And Doskey gives you the power to create *macro commands* to simplify complex DOS commands.

You load Doskey into memory with the Doskey command, which uses the following format:

doskey /reinstall /bufsize=*size* /insert /overstrike

/reinstall is an optional switch that reloads Doskey into memory. You would use this if you ran an MS-DOS session under Windows, DESQview, or some other multitasking program, and wanted to install Doskey for that command environment.

/bufsize=*size* sets the size of Doskey's internal buffer, in which it stores previous command lines and macros. *size* is the size of the buffer in bytes. When /bufsize isn't specified, Doskey uses a buffer size of 512 bytes, or 0.5K. If you were planning on writing several macros for Doskey and wanted a larger buffer, you could specify a 2K buffer by installing Doskey with:

doskey /bufsize=2048

/insert and */overstrike* determine whether Doskey starts in the insert mode or overstrike mode. Only one may be specified. The default is */overstrike*, meaning that new text you enter will overwrite existing text, as in a word processor's typeover mode. Specifying */insert* allows you to insert text into a command, which is how most word processors operate by default.

Doskey uses other options as well, each of which is discussed in the sections that follow. Normally you don't type any parameters when you first load Doskey into memory. In fact, accessing this power is as simple as putting the line *doskey* into your AUTOEXEC.BAT file. And since it uses precious conventional RAM, you might want to consider loading Doskey into an upper memory block using the Loadhigh command:

```
loadhigh doskey
```

Upper memory blocks and the Loadhigh command are covered in Chapter 9. Note that you need a 386 or later PC with extended memory and all the proper device drivers to accomplish this feat. Otherwise, Doskey takes up only a small sliver of memory, usually 4K or so. Nothing to lose sleep over.

Doskey's editing keys

Doskey gives you some powerful new keys to use when editing a command line:

Key	*Function*
Left arrow	Moves the cursor left one character
Right arrow	Moves the cursor right one character
Ctrl-left arrow	Moves the cursor left one word
Ctrl-right arrow	Moves the cursor right one word
Home	Moves the cursor to the start of the command
End	Moves the cursor to the end of the command
Ctrl-End	Deletes from the cursor to the end of the command
Ctrl-Home	Deletes from the cursor to the start of the command
Del	Deletes the current character
Ins	Toggles insert/overwrite modes
Ctrl-T	Command separator

In addition to these command keys, all of DOS's regular function keys are available. (See Figure 13-1.) The Escape key, however, will not display a backslash and move down to the next line; instead, pressing Escape erases the current line. Otherwise, the function keys behave as they normally do when Doskey hasn't been installed.

To try these new editing commands, install Doskey by entering *doskey* at the DOS prompt if you haven't already done so. You'll see

```
C>doskey
DOSKey installed.

C>
```

If you don't see anything, Doskey is already installed. (It won't load twice unless you use the */reinstall* switch.) You're now ready to use the new command-line editing keys. First change to your DOS subdirectory:

```
C>cd dos
```

Now list all EXE files sorted alphabetically—just as was done earlier in this chapter. Type the following command:

```
C>dir *.exe ¦ sort
```

Press Enter to see the sorted files. Now edit the command. Press F3 to redisplay it. Press the left-arrow key 10 times to back up to the EXE part of the command.

With Doskey installed, the left-arrow key won't back up and erase. Press the End key—you're at the end of the line. Now press the Ctrl-left arrow key combination three times. See how you hop from word to word within the command?

You can use the Ins and Del keys for editing as well. Position the cursor over the first E in EXE. Press Del three times. Now press the Ins key to turn on the insert mode. The cursor changes from an underline to a half-block, indicating that you're in the insert mode. Type *com*. The command should now look like this:

```
C>dir *.com ¦ sort
```

Press Enter to see the sorted COM files. Note that you didn't have to press F3 or the End key to move to the end of the command; the cursor can be anywhere on the command line, and DOS will accept it.

Press F3 to redisplay the command line a third time. You're now going to edit the line to display all the SYS files in the \DOS subdirectory. Press the Home key to move the cursor to the start of the line. Now press F2, the search key, and a *C*. The cursor is positioned right under the C in COM. Type *sys*.

Because the overstrike mode is active, SYS replaces COM in the command line. Press Enter to see a sorted directory of all your SYS files.

Because you're using DOS 5, you needn't include the *sort* filter in a Dir command. Press F3 again to redisplay the command line. Press Ctrl-left arrow twice to back up to the pipe character (¦). Now press Ctrl-End to delete the rest of the line. Type */o*. The final line should look like this:

```
C>dir *.sys /o
```

Press Enter to see the list of sorted SYS files. The Ctrl-Home key command also works like Ctrl-End, but it deletes from the cursor's position to the beginning of the command line.

This editing is powerful and quick. Combined with the original DOS command-line editing keys, Doskey really speeds up editing DOS commands. Anyone who's used to a word processor will immediately take to the new editing keys' logic.

The command separator

When Doskey is loaded, you get another command-line bonus: The command-separator character, Ctrl-T. Using the command separator, you can put multiple DOS commands on one command line. The commands will execute one after another, just as if they were entered into a batch file.

With Doskey loaded, type the following command line. The ¶ character is produced by pressing Ctrl-T:

```
C>cd \ cls ¶ dir
```

Press Enter and you'll see each command being issued separately at the DOS prompt: First you see *cd * to move to the root directory; then *cls*, which clears the screen; and finally the *dir* command. It's as if you stored all three commands in a secret batch file and had DOS execute them one after the other. That's the benefit of the command-line separator, Ctrl-T.

For more visual feedback, enter the following:

```
C>ver ¶ vol ¶ cd
```

You'll see each command as it's issued at its own DOS prompt, complete with its own feedback; it doesn't work like a batch file, but it's just as quick. The spaces around the ¶ character are optional.

Unlike a batch file, multiple DOS commands on a command line don't give you any programming power—and you can't turn off the display with an *echo off*. It's just a convenient way to issue more than one command at a time, just as long as you separate each with Ctrl-T.

A major drawback to this technique is that it won't work at all in a batch file. For example, putting the line *cd \ ¶ cls ¶* dir in a batch file will generate an error; the batch-file interpreter waltzes around the Doskey program in memory (even though it passes each line in the batch file to DOS, just as if you typed that line at the DOS prompt).

Also, note that pressing Ctrl-C to cancel multiple commands on the same line works only once for each command. DOS keeps executing the multiple commands, canceling only the current command each time you press Ctrl-C.

The Command History

As you enter each command at the DOS prompt, Doskey stores it in memory. You can recall those commands using the arrow keys, after which you can edit the commands or simply reissue them by pressing the Enter key.

To see a list of the commands you've already issued at the DOS prompt, you use Doskey's */history* switch. The format is:

doskey /history

/history causes Doskey to display a list of all the commands you've entered recently, or at least the commands that still fit into the buffer. The abbreviated form for */history* is */h*.

After entering the *doskey /history* command, you'll see something like the following on your display:

```
C>doskey /history
cd \dos
dir *.exe ¦ sort
dir *.com ¦ sort
dir *.sys ¦ sort
dir *.sys /o
cd \ ¶ cls ¶ dir
cd ¶ cd ¶ cd
ver ¶ vol ¶ cd
ver ¶ vol ¶ cd
doskey /history
```

Those are all your recent DOS commands as stored in Doskey's internal buffer. They're listed in order from oldest to most recent. And the best part is that you can recall any one of those commands, using the following keys.

Key	*Function*
Up arrow	Recalls previous command line
Down arrow	Recalls next command line (in the list)
PgUp	Recalls top command line (in the list)
PgDn	Recalls the last command line issued
F7	Lists all command lines
Alt-F7	Erases all command lines
F8	Command-line match/text search
F9	Selects command line by number

Press the up-arrow key. You'll see the previous command displayed. At first, this works just like pressing F3. But keep pressing the up-arrow key. Successively, each command you have entered is displayed from the most recent to the oldest. Pressing the down-arrow key moves through the commands from the oldest to most recent, given your current position in the list. The PgUp and PgDn keys are used to move to the top and bottom of the list.

As you recall old command lines from the list, note that you can edit each command and then press Enter to carry it out. This has two benefits: It cuts down on typing, and it lets you see which commands you issued, which helps you remember what you've done in your current DOS session.

Although the */history* switch displays your previous commands, it doesn't help you get at them. Sure, you can press the up-arrow key to find a command. But there are quicker ways to do it.

The F7, F8, and F9 keys can be used to quickly fetch a command from the history list. F7 displays the command history just like *doskey /h*, but it adds numbers to the commands displayed, as well as its own *more* prompt if the list is long. After you press F7, you'll see a display similar to this:

```
1: cd \dos
2: dir *.exe ¦ sort
3: dir *.com ¦ sort
4: dir *.sys ¦ sort
5: cd \ ¶ cls ¶ dir
6: cd ¶ cd ¶ cd
7: ver ¶ vol ¶ cd
8: ver ¶ vol ¶ cd
9: doskey /history
C>
```

By itself, this display is only moderately useful. But with the F9 key, you can use the numbers to snatch a specific command from the list. For

example, if you want to see the list of sorted EXE files again, press F9. You'll see:

```
C> Line number:
```

From the list displayed by F7, type the proper number, *2*. Press Enter. You'll see:

```
C>dir *.exe ¦ sort
```

Note that Doskey only displayed the command; it didn't execute it. This gives you an additional opportunity to edit the command or to cancel it by pressing Escape.

A more powerful key is F8, which searches through the list to match whichever keys you've entered. For example, entering *di* at the DOS prompt and pressing F8 displays the most recent command starting with *di*. Pressing F8 a second time displays the next most recent command, and so on.

F8 works like a quick-search routine in a word processor or database. For example, if you want to reissue the most recent Copy command, just type *copy*, press F8, and press Enter. The latest Copy command will be displayed on the command line—press Enter to carry it out again.

The last Doskey function key command that deals with the command-history list is Alt-F7, which clears the command history, erasing all the entries. This can be useful in many circumstances, including erasing previous commands to keep any snooping users from seeing what you're doing (and noticing passwords and hidden directories). And it can also be used when creating batch files using Doskey.

Creating batch files using Doskey

A batch file need be nothing more than a collection of DOS commands. Since Doskey keeps track of your DOS commands, you can use its history function to assist in the creation of a batch file. This allows you to enter and test/debug the DOS commands as you go.

To start creating a batch file using Doskey, clear the command-history list by pressing Alt-F7. Next enter the key commands in your batch file. For example, enter the following four DOS commands:

```
@echo off
echo Subdirectory status of:
cd
dir ¦ find "file(s)"
```

The first command, *@echo off*, will produce a *Bad command or file name* error. That's okay. The other commands will carry out their functions properly. If you make a mistake, reedit the command line using the Doskey editing keys. Don't worry about starting over; there will be editing required anyway.

After you've completed entering the commands and made sure they work, it's time to create the batch file.

Pressing F7 displays a numbered list of the commands—but it doesn't let you use I/O redirection to create a file. So to create a batch file, you use the *doskey /history* command with I/O redirection. Enter the following:

```
C>doskey /history > dirstat.bat
```

Use your word processor, text editor, Editor, or Edlin to load and edit the file created, DIRSTAT.BAT. It should look like this:

```
@echo off
echo Subdirectory status of:
cd
dir ¦ find "file(s)"
doskey /history > dirstat.bat
```

Delete the final line, where the command history was redirected to the file DIRSTAT.BAT. A command becomes a part of the history list as soon as you press Enter, so there's no way to avoid inclusion of that line in the list.

When you're done editing, save the file and exit your word processor or text editor. Now you have a batch file, DIRSTAT.BAT, which displays the current subdirectory, number of files, and amount of disk space used. All the commands have been tested using Doskey:

```
C>dirstat
Subdirectory status of:
C:\
       16 file(s)      58024 bytes
```

More complex and involved batch files can be created this way. But you'll still need to edit out that final line, as well as add labels and batch-file commands as needed. Remember to press Alt-F7 before you start "recording." Also, note that the command separator, ¶, will work when you record a command but will cause an error if it's transferred to and executed by a batch file.

Creating Doskey Macros

And if everything Doskey does so far isn't enough, the program also allows you to create macros for DOS commands and use them as shortcuts. A macro isn't the same as a batch file. It's a shortcut: a word or a combination of one or more keys that represents a longer DOS command. You can assign one or more DOS commands to a single macro word and use that word to play back the commands assigned to it.

To create a Doskey macro, you must first install Doskey itself. You then use Doskey's *macro* option, followed by an equal sign and the complete command line:

doskey *macro=command*

macro is the name of the Doskey macro. It can be from one to several characters long, but it cannot contain spaces or other forbidden characters. It's best to use a short yet descriptive name for *macro*.

command is a DOS command-line command. Doskey assigns that command to the named macro. When you enter the name of the macro at the command prompt and press Enter, DOS will display the assigned command and execute it. If you omit *command*, the macro is erased. (This works like the Set command does when it erases an environment variable; see Chapter 8.)

Doskey's macros can be quite easy or quite complex. They are similar to batch files in that a macro is a small word that represents a larger command or, if you use the ¶ (Ctrl-T) command separator, many DOS commands. But a macro isn't a batch file, and in fact, you cannot use a Doskey macro inside a batch file.

A simple macro to create is the D macro, which replaces the command *dir /w*. To make this macro, make sure Doskey is already loaded, and enter

```
C>doskey d=dir /w
```

D is the macro's name, and its command is *dir /w*. The macro is now recorded in memory and ready to use. Type *d* at the command prompt and press Enter to see it work:

```
C>d
C>dir /w
```

After you enter the macro, a second DOS prompt appears, and the command assigned to that macro is issued. Since the D macro is now part of

memory, you can use it anywhere, in any subdirectory. Unlike a batch file, which must be in the current directory or a directory on the search path, a macro is available at any DOS prompt.

Here is another handy macro. Enter the following:

```
C>doskey h=doskey /history
```

Now the H macro is assigned to *doskey /history*. Entering H on the command line will give you a command-history list.

To see the macros you've created, use Doskey's */macros* switch, which can also be abbreviated as */m*:

```
C>doskey /macros
D=dir /w
H=doskey /history
```

Like the Set command, Doskey converts the first part (the macro name) to uppercase, whereas the command remains in whichever case you entered. Also, as with the Set command, to remove a macro you simply specify an equal sign followed by nothing:

```
C>doskey d=
```

This command erases the D macro from the list.

Note that macros must always represent full DOS commands. You cannot create a partial macro for use with other commands (which is how some application programs' macros work). For example, consider the following macro:

```
C>doskey all=*.*
```

The ALL macro now represents *.*, all files in the directory. You might think you could use it as a part of a DOS command, as in:

```
C>dir all
```

But this doesn't work. Doskey translates a macro only when it appears as the first element in the command line. The above command is really *dir all*, which displays a directory of any files named ALL.

There is one function key that can be used with Doskey macros, Alt-F10. Alt-F10 deletes all macros in memory, just as Alt-F7 deletes the command history.

Using Doskey's special characters

Doskey uses special characters that you can include as part of a macro command:

Macro Command	*Represents*
$B	The pipe character (¦)
$G	The greater-than character (>)
$T	The Ctrl-T character command separator (¶)
$$	The dollar-sign character ($)
$1 through $9	Command-line parameters
$*	All command-line parameters

Note that some of these special characters are similar to the Prompt command's special characters as well as to the replaceable parameters used in batch files. They have similar meanings when creating macros.

The first special characters are $B for the pipe character, $G for greater than, $L for less than, and $T for the command-separator character. These are necessary when creating a macro because otherwise DOS would interpret them as commands. For example, consider the following macro:

```
C>doskey ds= ¦ sort /r ¦ more
```

This command creates a macro *ds*, assigning to it the command *dir ¦ sort /r ¦ more*. Or does it? Actually, DOS will interpret those pipes as immediate commands. The intended result won't be accomplished. Instead, use the special character $B to represent the pipe character:

```
C>doskey ds=dir $B sort /r $B more
```

Now the command will work, and the proper macro will be created. The same holds true for I/O redirection (< and >) and using the command separator (Ctrl-T). You must use the special dollar-sign commands to represent the characters, or DOS will misinterpret your intentions. And, as with the Prompt command, $$ is used to represent the dollar sign character itself. Don't let that one throw you.

Other special characters are used to indicate replaceable parameters that can be typed after the macro on the command line. Like a batch file's replaceable parameters, *%1* through *%9*, the special characters $1 through $9 are used to represent the first nine items typed after the macro's name on the command line. A tenth special character, $*, represents the entire line.

As an example, enter the following macro definition.

```
C>doskey scan=dir $1 /s $g \dscan $t type \dscan $b more $t del \dscan
```

This macro will look for all files matching whatever you type after *scan* on the command line. It actually contains three DOS commands, each separated by the *$t* command separator:

```
dir $1 /s > \dscan

type \dscan ¦ more

del \dscan
```

In the first command, the Dir command is issued followed by the first replaceable parameter. This allows you to enter the macro as *scan *.bat*, for example, to list all batch files in the current subdirectory and in all subdirectories underneath it (which is what the /s switch does). The output is redirected to the file \DSCAN. Note how the $G special character is used instead of >.

In the second command, the file \DSCAN is typed, and its output piped through the More filter. Note how $B replaces the pipe character (¦) in the macro definition.

Finally, \DSCAN is deleted.

Enter the macro and run it. If you make a mistake, don't worry; because Doskey is loaded, you can always recall that command line and reedit the macro. (This is the way all macros are edited once they've been entered.)

Note how $T was used to represent the ¶ command separator. In the end, *scan* is a powerful macro representing three DOS commands. However, the display isn't as clean as with a batch file; there is no *echo off* command for a Doskey macro.

There are a few drawbacks to creating macros with Doskey. Your macros will not work in batch files because batch files execute DOS commands in a way that circumvents Doskey. Also, creating macros decreases the size of your Doskey command history.

DOS commands and program names will run no matter how many spaces you put before them, but a Doskey macro will run only when it's the first item after the DOS prompt. You can turn this quirk to your advantage by giving a macro the same name as a DOS command. Try the following:

```
C>doskey dir=dir $1 /o
```

dir is now a macro for the Dir command, complete with a replaceable parameter and the sort switch set to sort by file name. You can do this with any DOS command or the name of any program, and it can protect your files from some of the deadlier DOS commands. For example, you can write macros called *del* and *format*, assigning them to safety-checking batch files that later run the real Del and Format commands.

To run the true DOS command, place one or more spaces before the command name. For example:

```
C>  dir
```

Now the "real" Dir command will run, not the macro.

Doskey macros must be created one at a time, and each time you create a macro, you must use the format *doskey macro=command*. However, you can create a batch file to do this for you once the macros have been written:

```
C>doskey /macros > makemac.bat
```

This batch file redirects the output of the *doskey /macros* command to the file MAKEMAC.BAT. Using your word processor or text editor, you can then insert the necessary Doskey command in front of each line of text, then top it off with an initial *@echo off* command. Ta-da, an instant macro-creating batch file! Ideally, you could incorporate the list into your AUTOEXEC.BAT file after you initially load Doskey.

Redefining the Keyboard

The application programs you use probably assign special meanings to many or all of the function keys, the keypad keys, perhaps combinations that use the Alt and Ctrl keys, and sometimes even the character keys. DOS gives special meaning to some keys, such as the editing keys described in the previous section and F6 to mark the end of a file; and combinations with the Ctrl key such as Ctrl-C to cancel a command and Ctrl-S to stop the display.

The ANSI.SYS Define Key command, described in Chapter 3, gives you the same sort of control over the keyboard. You can not only change the character a key produces, you can also assign a string of characters to a single key. As you'll see in a moment, this presents some real possibilities.

As in Chapter 3, the Prompt command is used to demonstrate this ANSI.SYS command. If you haven't yet gone through the examples in Chapter 3, you might do so before continuing here.

Assigning a Command to a Key

If you frequently use a particular command, especially a long one, it would be much more convenient to type it with a single keystroke. It's an obvious advantage to press just one key to run *dir *.doc ¦ sort /r /25 ¦ more*. You could, of course, write a batch command and give the batch file a single-character name, or create a Doskey macro if you have DOS version 5. But you can save the space the batch file would occupy on the disk and the time required to load it by assigning the command to a key you normally don't use.

It's easy to demonstrate this with the Prompt command. Enter the following to assign *erase *.bak* to Alt-B:

```
C>prompt $e[0;48;"erase *.bak";13p
```

Now, when you press Alt-B, DOS displays *erase *.bak* as if you had typed it, erasing all files in the current directory with the extension BAK. This is a quick way to clean up directories with a lot of BAK files.

If you'd like this definition—or any other—to be in effect permanently, put the ANSI.SYS command in your AUTOEXEC.BAT file. You can use a Prompt command, as shown here, or an Echo command, as shown in Chapter 3. If you use a Prompt command in your AUTOEXEC.BAT file, be sure to define your system prompt *after* redefining the key.

You can assign only one command to a key. As soon as DOS sees a carriage return (character 13) in the string of characters produced by a key, it reacts as if you had pressed the Enter key. DOS checks to see whether what has been entered is a valid command, ignoring any characters that follow.

You can carry out more than one command by pressing a single key if you write a batch file that contains the commands and then assign the batch command to a key.

For example, suppose you use a system of menus to select application programs. If the name of the batch file that starts the menu system is MAINMENU.BAT and it's in a directory in the command path, you could assign the batch command to a key combination—Alt-F10, say—and start your menu system from any directory.

Changing the Keyboard Layout

You can redefine the keyboard any way you choose, but chances are you'll most often redefine the function keys or combinations with the Alt and Ctrl keys. There are times, however, when it could be helpful to redefine

other keys. The two batch files described here redefine the keypad keys that normally move the cursor:

- BOX.BAT assigns a set of box-drawing characters to the keypad, letting you type these characters in the extended character set directly from the keyboard.
- UNBOX.BAT reverses BOX.BAT, restoring the normal definition of the keypad keys.

The box-drawing characters are assigned to positions in the keypad that generally correspond to their true location in a box. Figure 13-2 shows the layout for the IBM PC, IBM PC/XT, and compatibles. Figure 13-3 shows the box-drawing characters for the IBM PC/AT and AT-compatible computers.

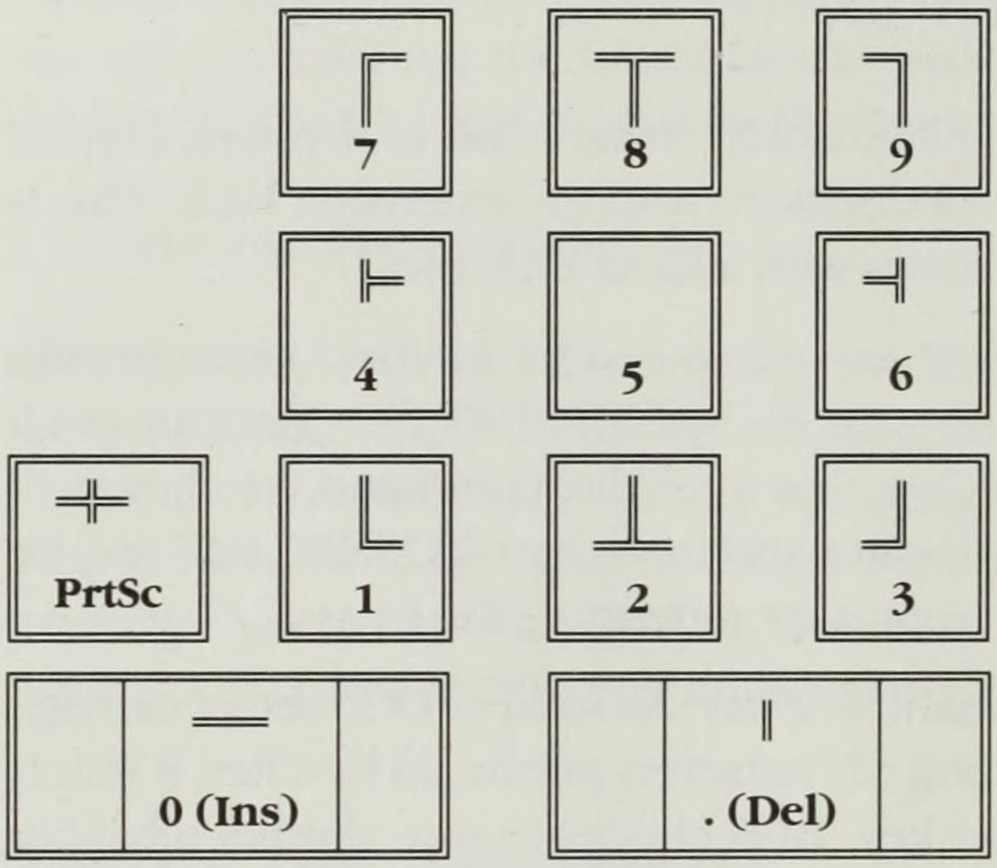


Figure 13-2. *Box-drawing characters on an IBM PC and an IBM PC/XT keypad.*

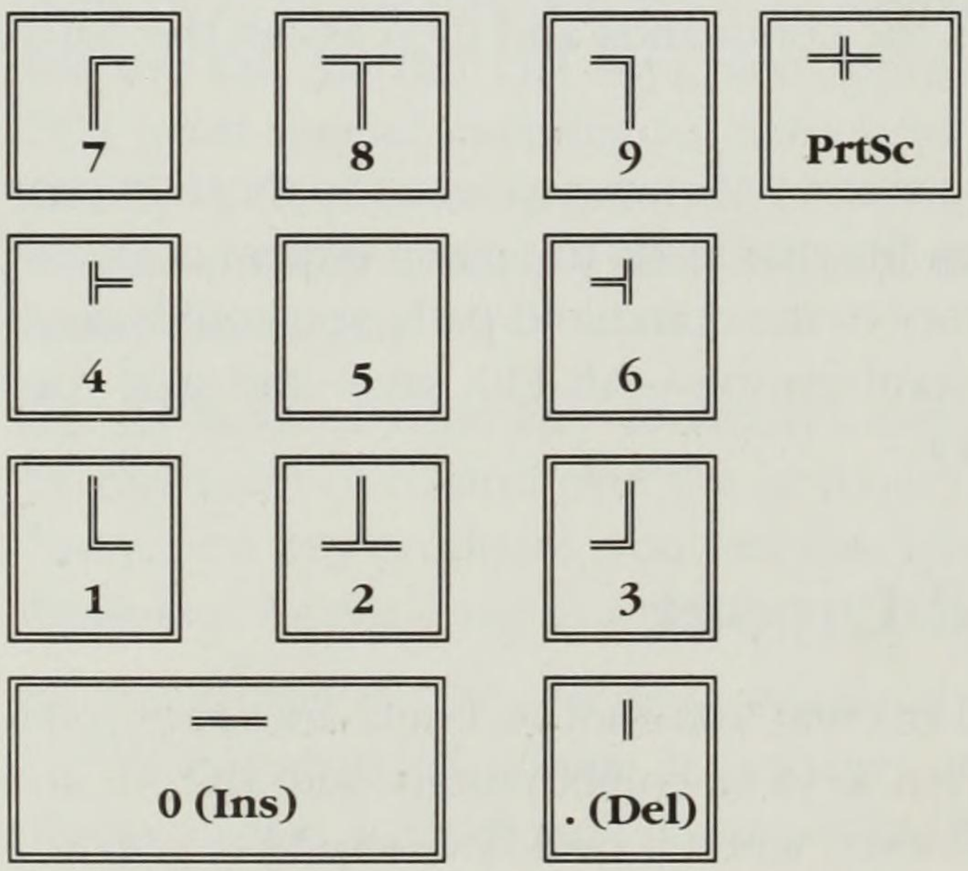


Figure 13-3. *Box-drawing characters on an IBM PC/AT keypad.*

On the IBM enhanced keyboard, PrtSc and asterisk (*) are assigned to their own keys, so use the asterisk located directly above PgUp on the numeric keypad.

Because UNBOX.BAT restores the original key definitions, you'll enter both batch files, then test them together.

Figure 13-4 shows the commands in BOX.BAT:

```
1: @echo off
2: echo [0;71;201p
3: echo [0;72;203p
4: echo [0;73;187p
5: echo [0;75;204p
6: echo [0;77;185p
7: echo [0;79;200p
8: echo [0;80;202p
9: echo [0;81;188p
10: echo [0;82;205p
11: echo [0;83;186p
12: echo [42;206p
13: cls
```

Figure 13-4. *BOX.BAT: Redefining the keypad keys to draw boxes.*

BOX.BAT is straightforward but not especially easy to read. Each Echo command following *@echo off* sends an ANSI.SYS Define Key command to the console. The first two numbers in all but the last Echo command in line 12 identify the key to be redefined (in the first command, for example, 0;71 is the Home key). Line 12 redefines the asterisk (*), whose key code is 42, because the PrtSc key has the same key code. Appendix D contains the complete list of key codes.

The Define Key command lets you define more than one key, so you could use just one Echo command for all 11 definitions in Figure 13-4:

```
echo[0;71;201;0;72;203;0;73;187;0;75;204;0;77;185;0;79;200;0;
80;202;0;81;188;0;82;205;0;83;186;42;206p
```

But this would make BOX.BAT even harder to read and make changing it almost more trouble than it's worth. The minimum disk space required by a file is much larger than BOX.BAT, so there's no advantage to combining all of the Define Key commands into one Echo command.

UNBOX.BAT is quite similar to BOX.BAT; it restores each of the 11 keypad keys redefined by BOX.BAT to their original meaning. Figure 13-5 shows the commands in UNBOX.BAT.

```
 1: @echo off
 2: echo [0;71;0;71p
 3: echo [0;72;0;72p
 4: echo [0;73;0;73p
 5: echo [0;75;0;75p
 6: echo [0;77;0;77p
 7: echo [0;79;0;79p
 8: echo [0;80;0;80p
 9: echo [0;81;0;81p
10: echo [0;82;0;82p
11: echo [0;83;0;83p
12: echo [42;42p
13: cls
```

Figure 13-5. *UNBOX.BAT: Restoring the keypad keys' definitions.*

Using BOX.BAT and UNBOX.BAT

Create BOX.BAT and UNBOX.BAT with your text editor or word processor, and enter the commands shown in Figures 13-4 and 13-5. To put the box-drawing characters on the keypad, type the BOX.BAT command:

```
C>box
```

BOX.BAT echoes 11 blank lines, one for each Echo command that follows *@echo off*, then clears the screen.

Now try the keypad keys. DOS should display the box-drawing characters. If you press Enter, of course, DOS should respond *Bad command or file name*. Notice that the Ins key doesn't repeat. This has nothing to do with BOX.BAT; the Ins key normally never repeats.

To restore the original definitions for the keypad keys, enter:

```
C>unbox
```

UNBOX.BAT, too, echoes 11 blank lines and clears the screen. The keypad keys should have returned to their original meaning.

You could write other batch files that use this technique to put other characters from the extended character set (such as the mathematics symbols or foreign-language characters) temporarily on the keyboard.

To create a file using the box-drawing characters, copy from the console or use one of the batch commands for creating a text file (described in Chapter 14). You can't use your text editor or word processor with the

keypad keys redefined. Even if the program doesn't define its own meanings for the keypad characters, you wouldn't be able to move the cursor.

The IBM extended character set includes four sets of box-drawing characters that use different combinations of single and double lines. Figure C-3 in Appendix C shows all four sets. You could expand BOX.BAT to add the other three sets in different parts of the keyboard, or write four different versions (perhaps called BOX1.BAT through BOX4.BAT) to put any set on the numeric keypad. If you used four different batch files to redefine the keypad keys this way, UNBOX.BAT would restore the numeric keypad after you used any of them.

Speeding Up the IBM PC/AT and PS/2 Keyboards

When you hold down a key, the keyboard waits a moment, then repeats the character until you release the key (or until the keyboard buffer is filled). Both the length of the delay and the speed of the repeat action can be changed on the IBM PC/AT, PS/2, and most compatible computers.

Using the Mode command

Beginning with version 4 of DOS, you can adjust the speed and delay rates of your keyboard using the Mode command in the following format:

mode con rate=*speed* delay=*pause*

con is the name of the console device, the keyboard in this instance.

rate=*speed* sets the speed at which DOS repeats the keystroke. Values range from 1 through 32, with 1 repeating the character approximately twice per second, and 32 repeating approximately 30 times per second.

delay=*pause* specifies how long DOS waits before it starts repeating the key. Values for *pause* range from 1 for ¼ second through 4 for a full second.

You must specify both *speed* and *pause* values with the Mode command. For example, *mode con rate=28 delay=2* sets the keyboard to a repeat rate of 20 characters per second with a delay of ½ second—about average for most PCs. To have the keyboard respond more quickly, use the following Mode command:

```
C>mode con rate=32 delay=1
```

Now typing really zips along! If that's too fast, try a slower version of the command:

```
C>mode con rate=1 delay=4
```

That's almost agonizingly slow, so you may want to reset the PC back to more-or-less normal with *mode con rate=28 delay=2*.

Creating FAST.COM

The program FAST.COM reduces the delay time and increases the speed of the repeat action and is especially useful if you are using a version of DOS earlier than 4. The program is specific to the IBM PC/AT, PS/2, and compatible computers. You cannot use it with the IBM PC, IBM PC/XT, or computers compatible with those models.

You create FAST.COM using Debug. The procedure is described in Chapter 5. If you haven't yet gone through the examples in Chapter 5, you might do so before continuing here.

Using your text editor, word processor, Editor, or Edlin, create a file named FAST.SCR and enter the lines shown in Figure 13-6.

Proofread FAST.SCR against Figure 13-6. Make certain that line 9 is blank and that the last line is a *q*. If there is an error in either of these lines, you'll have to restart DOS when you try to create FAST.COM. When you're sure there are no differences, save FAST.SCR.

To create FAST.COM, enter the following Debug command:

```
C>debug < fast.scr
```

If nothing seems to happen and pressing a key has no effect, restart DOS, proofread FAST.SCR against Figure 13-6, correct any differences, and re-enter the Debug command.

```
1: a 100
2: mov  al,f3
3: out  60,al
4: mov  cx,2000
5: loop 107
6: xor  al,al
7: out  60,al
8: int  20
9:
10: r cx
11: f
12: n fast.com
13: w
14: q
```

Figure 13-6. *Debug script for FAST.COM.*

```
-a 100
xxxx:0100 mov  al,f3
xxxx:0102 out  60,al
xxxx:0104 mov  cx,2000
xxxx:0107 loop 107
xxxx:0109 xor  al,al
xxxx:010B out  60,al
xxxx:010D int  20
xxxx:010F
-r cx
CX 0000
:f
-n fast.com
-w
Writing 000F bytes
-q
```

Figure 13-7. *DOS response to creating FAST.SCR.*

DOS should respond as shown in Figure 13-7. Proofread your display against Figure 13-7. If there are any differences, proofread FAST.SCR against Figure 13-6, correct the differences, and reenter the Debug command.

If your screen agrees with Figure 13-7, you're ready to try FAST.COM. But before you enter the command, hold down any key and notice how quickly the key repeats. Now enter the Fast command:

```
C>fast
```

Hold down a key again; you should notice a significant difference. If you don't, go back and check everything again.

This increased speed should remain in effect even when you use an application program. FAST.COM can do a lot to reduce your impatience level when you must scroll all the way across a line or use the Backspace key to erase more than a few characters. Although the actual time saved may be small, comfort and satisfaction often depend as much on perceptions as on objective reality. It may take a while to get used to the increased speed, but it's not an unpleasant adaptation.

Turning Off Num Lock

The IBM enhanced keyboard includes both cursor-control keys and a numeric keypad. To maintain compatibility with previous keyboards, however, the keys on the numeric keypad still double as cursor-control

keys; this means you must turn on Num Lock to enter numbers with the numeric keypad. So that you don't have to press the Num Lock key, some systems automatically turn on Num Lock whenever you start or restart your computer.

But if you still use the keypad keys as cursor-control keys, you have to turn Num Lock off every time you start or restart your computer.

The short program described here—NUMOFF.COM—turns Num Lock, Scroll Lock, and Caps Lock off. Put it in AUTOEXEC.BAT, and it will turn Num Lock off automatically when you start or restart your computer.

You create NUMOFF.COM using Debug. The procedure is described in Chapter 5. If you haven't yet gone through the examples in Chapter 5, you might do so before continuing here.

Using your text editor, word processor, Editor, or Edlin, create a file named NUMOFF.SCR and enter the lines shown in Figure 13-8.

Proofread NUMOFF.SCR against Figure 13-8. Make sure that line 8 is blank and that the last line is a *q*. If there is an error in either of these lines, you'll have to restart DOS when you try to create NUMOFF.COM. When you're sure there are no differences, save NUMOFF.SCR.

```
 1: a 100
 2: mov  ax,40
 3: mov  ds,ax
 4: mov  ah,0
 5: mov  bx,0017
 6: mov  [bx],ah
 7: int  20
 8:
 9: r cx
10: e
11: n numoff.com
12: w
13: q
```

Figure 13-8. *Debug script for NUMOFF.COM.*

To create NUMOFF.COM, enter the following Debug command:

```
C>debug < numoff.scr
```

If nothing seems to happen and pressing a key has no effect, restart DOS, proofread NUMOFF.SCR against Figure 13-8, correct any differences, and re-enter the Debug command.

DOS should respond as shown in Figure 13-9. Proofread the display against Figure 13-9. If there are any differences, proofread NUMOFF.SCR against Figure 13-8, correct the differences, and reenter the Debug command.

```
-a 100
xxxx:0100 mov  ax,40
xxxx:0103 mov  ds,ax
xxxx:0105 mov  ah,0
xxxx:0107 mov  bx,0017
xxxx:010A mov  [bx],ah
xxxx:010C int  20
xxxx:010E
-r cx
CX 0000
:e
-n numoff.com
-w
Writing 000E bytes
-q
```

Figure 13-9. *DOS response to creating NUMOFF.SCR.*

If your screen agrees with Figure 13-9, you're ready to try NUMOFF.COM. First make sure the Num Lock light is *on* in the upper right corner of the keyboard; if it isn't, press the Num Lock key.

Now enter *numoff.* The Num Lock light should go out. If it doesn't, go back and check everything again.

NUMOFF.SCR doesn't do much; it just eliminates a minor inconvenience. But of such small things can large triumphs be made, such as making your computer work the way you want it to and not the way some well-meaning programmer believes it should.

Using Foreign-Language Keyboards

In Chapter 2, you learned a trick for entering foreign-language characters. In today's international society, that's often necessary. But remembering Alt-key combinations and extended ASCII codes can be a bother—especially when DOS provides you with several ways to easily access international characters.

DOS went international in version 3.0. Since then, many different commands, device drivers, and terms have taken on an international flavor. For example, *code pages* allow you to replace the standard IBM extended

ASCII set with special foreign character sets, and device drivers such as COUNTRY.SYS and KEYBOARD.SYS tell DOS about foreign currency symbols and date formats and how to sort under the rules of different alphabets. (These commands and their uses are covered in *Running MS-DOS*.) The Keyb command is used to reset your keyboard to use international characters.

The format of the Keyb command is:

keyb *country,codepage,KEYBOARD.SYS*

country is a two-letter country code that tells DOS which keyboard layout to use, depending on the country you specify. The codes are listed in Figure 13-10.

codepage is optional. It specifies a code page to use for a particular country. The codes differ depending on the country you select. They're listed in Figure 13-10. Note that although the *codepage* value is optional, you must follow *country* with two commas if you omit it.

KEYBOARD.SYS is the name of the keyboard information file that came with DOS. Unless the file is in the root directory, you need to specify a full path to KEYBOARD.SYS when you use the Keyb command.

If you don't specify any parameters, the Keyb command will display the keyboard currently in use:

```
C>keyb
KEYB has not been installed
Active code page note available from CON device
```

If you haven't installed a keyboard, the default U.S. keyboard is specified, which is one way to interpret the above message. To install a foreign keyboard, such as one for Latin America (Spanish), enter the following command:

```
C>keyb la,,c:\dos\keyboard.sys
```

Remember to specify a different path if KEYBOARD.SYS isn't located in your \DOS subdirectory. After entering the above command, you won't see any feedback. But your keyboard has now been reset to produce some special Spanish-language keys. For example, the semicolon key (;) now produces an ñ when pressed; the colon or shift-semicolon (:) produces a capital Ñ. The equals (=) and plus (+) keys now produce the ¿ and ¡ characters. Other keys on the keyboard will be in different positions; a copy of the Latin American keyboard layout is in the back of your DOS manual.

Country	*Code*	*Code Page*
Australia	us	437,850
Belgium	be	437,850
Canada-French	cf	863,850
Denmark	df	865,850
Finland	su	437,850
France	fr	437,850
Germany	gr	437,850
Italy	it	437,850
Latin America	la	437,850
Netherlands	nl	437,850
Norway	no	865,850
Portugal	po	860,850
Spain	sp	437,850
Sweden	sv	437,850
Swiss-French	sf	437,850
Swiss-German	sg	437,850
United Kingdom	uk	437,850
United States	us	437,850

Figure 13-10. *Each country has a two-letter code and valid code pages associated with the Keyb command.*

The Keyb command proves indispensable when you're writing in a foreign language. It works at the BIOS level in your computer, so even your word processor or spreadsheet program can take advantage of the new keyboard layout and characters. Still, it's easy to see how a keyboard that produces unusual characters may get in the way of other programs. To minimize this, there are three special key combinations you can use after the Keyb command to switch keyboards:

- Ctrl-Alt-F1, which restores the standard U.S. keyboard
- Ctrl-Alt-F2, which switches back to the installed keyboard driver
- Ctrl-Alt-F7, which activates a basic *typewriter* mode

You can switch back to your U.S. keyboard by pressing Ctrl-Alt-F1, then back to the Latin American keyboard (installed above) by pressing Ctrl-Alt-F2.

Once Keyb has been installed, you can use the Keyb command by itself to tell which keyboard you've installed:

```
C>keyb
Current keyboard code: LA  code page: 850
Active code page not available from CON device
```

Note that pressing Ctrl-Alt-F1 will not change this message.

Since the Keyb command is memory resident, it's possible to load it into high memory when you install it. For example, if you've set up your computer to use upper memory blocks, you can put the following command into your AUTOEXEC.BAT file:

```
loadhigh keyb fr,,c:\dos\keyboard.sys
```

The above command loads the French keyboard driver into the upper memory area using the Loadhigh command. Refer to Chapter 9 for more information on using the upper memory area, upper memory blocks, and the Loadhigh command.

Finally, beware of dead keys. Some international keyboards establish dead keys on the keyboard, which have no function and produce no character. So don't fear that your keyboard is broken if you press a key and it doesn't seem to work. Also, note how some keyboards will move the special DOS characters, specifically the backslash (\), to other keys. Refer to your DOS manual for their new location.

CHAPTER

14

Playing It Smart with Files and Disks

Most of your computer work involves creating, changing, printing, and otherwise dealing with files. The most common use of batch commands is working with files, which are stored on disks. This chapter suggests some ways of working with files and disks that can make your job a bit easier, and describes several batch commands designed to let you manage your files and disks more quickly, easily, and safely.

Naming Your Files

Naming your files in a consistent manner lets you manage them more easily. For example, if you always use an extension with file names but never with directory names, you can display just the subdirectories by typing *dir *.* at the DOS prompt. If you use the extension LET for all files that contain letters, you can display the names of all files that contain letters by typing *dir *.let*.

Suppose you followed this convention for naming files that contain letters:

- The extension is LET.
- The first three characters identify the recipient.
- The next two characters identify the month (01 through 12).
- The next two characters identify the day (01 through 31).

Now you could display the names of all files that contain letters by typing *dir *.let*, display the names of all letters written to HST by typing *dir hst**, or display the names of all letters written in July by typing *dir ???07**.

In general, the more accurately your file names and extensions describe file contents and the more consistent you are in assigning names, the more easily you can keep track of your files.

Setting Up Your Directories

Besides containing your application programs and data files for different management or project responsibilities, your directory system should provide a framework to support the ways you use your computer. Its structure should feel natural, and you should be able to find a particular file without searching through several similar-sounding directories.

Some batch files in this book suggest a few files to be kept in the root directory but, in general, keep this directory as free of files as possible. Reserve the root directory for subdirectories, COMMAND.COM, AUTOEXEC.BAT, CONFIG.SYS, and a minimum of other files.

Figure 14-1 shows four first-level subdirectories (in the root directory) that you could use to contain your most frequently used programs and batch files. You may prefer different directory names, but these are the sorts of files you want readily available yet easily distinguished. In addition to these, of course, you'll want directories for your major application programs, such as word processing and spreadsheets. All these directories should be in your command path.

Directory	*Description*
\DOS	For all DOS files (and nothing but DOS files). This makes it easy to find a DOS file or change to a new DOS version.
\BATCH	For batch files you use frequently, plus other files they require (for example, files for redirecting input such as CR.DAT, described in Chapter 6 and later in this chapter).
\PGM	For utility programs, programs used by your batch files, and application programs that don't require a separate directory.
\MENU	For all files required by your menu system (if you use one), including programs or batch files that you also use independently or that are used by other batch files. This makes it easy to keep track of your menu system or make a copy to transfer to another computer.

Figure 14-1. *Suggested subdirectories for programs and batch files.*

In addition to these subdirectories, DOS 5 and some applications could simply use a temporary first-level subdirectory called \TEMP or a RAM drive (although the name doesn't have to be \TEMP).

Preventing Accidental Changes and Deletions

The Attribute command lets you make a file read-only; once you have done this, the file can't be changed or deleted until you remove the read-only protection. One inadvertent change or deletion could make you wish you had done it.

It doesn't take long to protect the files that you don't change, or that you change infrequently, or those that simply must not be changed accidentally. If all the DOS files are in a directory named \DOS, for example, you can make them all read-only by typing *attrib +r \dos*.**.

Moving Around Through Directories

The .. entry in a directory listing represents the directory immediately above the current one. You can use it to move quickly around a directory structure. For example, if you're in the directory \123\BUDGETS\MKT:

- Type *cd ..* to change to \123\BUDGETS
- Type *cd ..\..* to change to \123
- Type *cd ..\mfg* to change to \123\BUDGETS\MFG

The Batch Files

The remainder of this chapter describes several batch files that make working with files and disks easier and, in some cases, safer.

The batch files are described in independent topics. Each topic includes a description of what the batch file does, a figure that shows the commands in the batch file, an explanation of how it works, suggestions for testing it after you have entered it, and, in some cases, suggestions about how you might better tailor it to your use.

Stop Typing Those Long Directory Names

GO.BAT changes to the directory that matches an *alias* you specify as the first parameter. If the second parameter is a file name (including wildcard characters), GO.BAT changes to that directory, clears the screen, and displays the directory entries that match the file name.

You can define as many aliases as you like in the batch file. Using short aliases can make changing directories much faster, especially if you have a directory structure several levels deep.

Figure 14-2 shows the commands in GO.BAT.

- Line 2 skips the instructions if you specified at least one parameter.
- Lines 4–8 display instructions for typing the Go batch command followed by a blank line.
- Lines 9–11 display the list of aliases and directories you have defined, then return to DOS. Line 10 defines *r* as the alias for the root directory; this isn't an especially useful alias, but the Echo command is included here as a model for the Echo commands you will add that describe your aliases, which should start here.
- Line 13 starts the list of If commands that check *%1* against the aliases you have defined. Again, the check for *r* as an alias is included here only as a model for the If commands that check for your aliases, which should start here. If *%1* matches an alias, GO.BAT changes to that directory.
- Line 14 returns to DOS if you didn't enter a second parameter.

- Lines 15 and 16 clear the screen and display the directory entries that match the file name you specified as the second parameter.

```
 1:   @echo off
 2:   if not "%1"=="" goto OK
 3:   cls
 4:   echo [1mCommand                Result[m
 5:   echo go {alias}               Change to directory defined for {alias}
 6:   echo go {alias} {filename}    Change directory and display
 7:   echo                          entries for {filename}.
 8:   echo <Alt-255>
 9:   echo [1mAlias                  Directory[m
10:   echo  r                       root
11:   goto END
12: :OK
13:   if %1==r cd \
14:   if "%2"=="" goto END
15:   cls
16:   dir %2
17: :END
```

Figure 14-2. *GO.BAT: Changing directories quickly.*

Testing GO.BAT

GO.BAT requires no other files. First, enter the command with no parameters:

```
C>go
```

GO.BAT should display its instructions, followed by a list of the aliases and directories you have defined:

```
Command                 Result
go {alias}              Change to directory defined for {alias}
go {alias} {filename}   Change directory and display
                        entries for {filename}.


Alias                   Directory
r                       root
```

Remember, the alias defined here for the root is to give you a model for the Echo and If commands you will add to GO.BAT to define your aliases and directories. To make sure GO.BAT is working, change to the root directory by entering:

```
C>go r
```

If your prompt doesn't show you the current directory, you can type a Change Directory (*cd*) or Directory (*dir*) command to verify that you have changed directories.

If you're in the root directory, you know that the Change Directory (*cd*) command in GO.BAT is working correctly. Test the last feature by entering the command again, but this time add a file name as the second parameter:

```
C>go r *.bat
```

GO.BAT should respond by displaying the directory entries for all the batch files in the root directory.

It's hard to appreciate the convenience of GO.BAT when the only alias is for the root directory, which already has an alias (the \). To tailor GO.BAT to your directories, you insert an Echo command for each alias in place of line 10 in Figure 14-2, and an If command for each alias in place of line 13. Suppose you wanted to define the following aliases for the associated directories:

Alias	***Directory***
mem	\word\corresp
bgt	\123\budgets
p	\telecomm\xtalk\portflio

You would enter the following Echo commands in place of line 10 in Figure 14-2 (line up the aliases and their directories under the Alias and Directory headings, as shown in Figure 14-2):

```
echo  mem          \word\corresp
echo  bgt          \123\budgets
echo  p            \telecomm\xtalk\portflio
```

You would enter the following If commands in place of line 13:

```
if %1==mem cd \word\corresp
if %1==bgt cd \123\budgets
if %1==p   cd \telecomm\xtalk\portflio
```

Pick the directories and aliases you want to use with GO.BAT, modify the batch file, and stop typing those long directory names and backslashes.

This aliasing technique is interesting and quite versatile. But as an astute DOS 5 user, you should remember that the Doskey command lets you create macros that will work much faster and do more than these

aliases—and they're easy to update and display using Doskey. For example, to create an R alias to log to your root directory, you enter:

```
C>doskey r=cd \
```

Here you don't even have to type *go r*. Just type *r* and Enter and you're there. Refer to Chapter 13, "Increasing Your Keyboard's IQ," for more information on the Doskey command and its macros.

Creating Text Files

The next four batch files let you create text files quickly without using a text editor or word processor. All four use the technique of copying from the console, but each is tailored to a specific use. Because you can't edit what you have entered, these batch files are best suited to short files, although you can use some of them to add to an existing text file.

Creating a Text File Quickly

FILE.BAT clears the screen and displays a ruler that marks the columns. It's a quick way to create a short text file for any use, such as small batch files.

Figure 14-3 shows the commands in FILE.BAT.

```
 1:    @echo off
 2:    if not "%1"=="" goto OK
 3:    echo [7mYou must specify a file name or prn.[m
 4:    echo Please re-enter the command.
 5:    goto END
 6: :OK
 7:    if %1==prn goto START
 8:    if not exist %1 goto START
 9:    echo <Alt-7>[7m%1 exists.[m
10:    echo Press F1 to replace it, any other key to quit.
11:    reply
12:    if errorlevel 59 if not errorlevel 60 goto START
13:    goto END
14: :START
15:    cls
16:    echo [7mEditing %1 - Press {F6}-{ENTER} after the last line.[m
17:    echo     5   10   15   20   25   30   35   40   45   50   55   60   65   70   75
```

Figure 14-3. *FILE.BAT: Creating a text file quickly.* *(continued)*

Figure 14-3. *continued*

```
18:    echo ╒═══╤═══╤═══╤
```

FILE.BAT warns you that you must specify either a file name or *prn* (for the printer) and returns to DOS:

```
You must specify a file name or prn.
Please re-enter the command.
```

Enter the command again, this time specifying FILETEST.DOC:

```
C>file filetest.doc
```

FILE.BAT responds by clearing the screen and displaying the instructions and ruler:

```
Editing filetest.doc - Press {F6}-{ENTER} after the last line.

    5   10   15   20   25   30   35   40   45   50   55   60   65   70   75
|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
```

Now you can type the file. Enter:

```
WORDS TO LIVE BY

When all else fails
Use bloody great nails.
^Z
```

Now FILE.BAT gives you the option of printing the file:

```
Filetest.doc created.
Press F1 to print it, any other key to quit.
```

Make sure your printer is turned on, and press F1. FILE.BAT sends the file to the printer and returns to DOS.

Now see what happens if you use FILE.BAT to create a file that already exists by entering:

```
C>file filetest.doc
```

FILE.BAT beeps, warns you that the file exists, and gives you the option of replacing the file or quitting:

```
Filetest.doc exists.
Press F1 to replace it, any other key to quit.
```

Press the Spacebar or any other key to quit.

The batch file BOX.BAT, described in Chapter 12, redefines the numeric-keypad characters to produce the box-drawing characters from the extended character set. If you enter and execute BOX.BAT before FILE.BAT, you can use the numeric keypad to easily create a text file that contains simple graphic figures, such as an organization chart—or to doodle if you're bored talking on the phone.

Adding to the Beginning or End of a File

ADD.BAT lets you add lines to the beginning or end of a text file. It requires two parameters: the name of the file to be changed, and *beg* or *end* to specify where to add the lines.

Figure 14-4 shows the commands in ADD.BAT.

```
 1:    @echo off
 2:    if not "%1"=="" goto ONE_PARM
 3: :INSTRUCT
 4:    echo <Alt-255>
 5:    echo [1mCommand                Result[m
 6:    echo add {file name} beg   Add to the beginning of {file}.
 7:    echo add {file name} end   Add to the end of {file}.
 8:    goto END
 9: :ONE_PARM
10:    if "%2"=="" goto INSTRUCT
11:    if %2==beg goto TWO_PRMS
12:    if %2==end goto TWO_PRMS
13:    goto INSTRUCT
14: :TWO_PRMS
15:    if exist %1 goto OK
16:    echo [7mI can't find a file named %1.[m
17:    goto INSTRUCT
18: :OK
19:    echo <Alt-255>
20:    echo Type the line or lines to be added.
21:    echo [7mAfter the last line, press {F6} and {ENTER}.[m
22:    echo <Alt-255>
23:    if %2==beg copy con+%1 add.$$$ > nul
24:    if %2==end copy %1+con add.$$$ > nul
25:    erase %1
26:    rename add.$$$ %1
27:    echo <Alt-255>
28:    echo [7mNew contents of %1:[m
29:    echo <Alt-255>
30:    more < %1
31: :END
```

Figure 14-4. *ADD.BAT: Adding to the beginning or end of a file.*

The following explanations describe how ADD.BAT works:

- Line 2 skips displaying the instructions if you specify at least one parameter.
- Lines 3–8 display instructions for entering the Add batch command, then return to DOS.
- Line 10 displays the instructions and returns to DOS if you specified only one parameter.
- Lines 11–13 display the instructions and return to DOS if you didn't specify *beg* or *end* as the second parameter.
- Line 15 skips the following error message if the file you specified exists.
- Lines 16 and 17 display an error message and command instructions, then return to DOS if the file you specified doesn't exist.
- Lines 19–22 display a blank line, instructions for how to proceed, and another blank line.
- Line 23 copies the console, then the file you named, to a temporary file named ADD.$$$ if you specified *beg*.
- Line 24 copies the file you named, then the console, to a temporary file named ADD.$$$ if you specified *end*.
- Lines 25 and 26 erase the file you named and change the name of the temporary file from ADD.$$$ to the file you named.
- Lines 27–30 display a blank line, a title line, another blank line, and the new contents of the file.

Testing ADD.BAT

ADD.BAT requires no other files. First enter the command with no parameters:

```
C>add
```

ADD.BAT should display the following instructions and return to DOS:

```
Command                 Result
add {file name} beg     Add to the beginning of {file}.
add {file name} end     Add to the end of {file}.
```

Enter the command again, this time specifying the file ADDTEST.DOC and *beg* to add lines to the beginning of the file:

```
C>add addtest.doc beg
```

ADD.BAT warns you that it can't find the file, redisplays the instructions, and returns to DOS.

```
I can't find a file named addtest.doc.

Command                 Result
add {file name} beg     Add to the beginning of {file}.
add {file name} end     Add to the end of {file}.
```

Enter the following to create the test file:

```
C>copy con addtest.doc
"Too much of a good thing
^Z
```

Now enter the command naming ADDTEST.DOC again; this time, ADD.BAT prompts you to enter the lines to be added:

```
C>add addtest.doc beg

Type the line or lines to be added.
After the last line, press {F6} and {ENTER}.
```

Enter the following to add a couple of lines:

```
Someone once said
(it must have been Mae West)
^Z
```

ADD.BAT shows you the result of adding the lines and returns to DOS:

```
New contents of addtest.doc:

Someone once said
(it must have been Mae West)
"Too much of a good thing
```

Adding to the end of a file works the same way; enter the following:

```
C>add addtest.doc end

Type the line or lines to be added.
After the last line, press {F6} and {ENTER}.

is wonderful."
^Z

New contents of addtest.doc:
```

```
Someone once said
(it must have been Mae West)
"Too much of a good thing
is wonderful."
```

Creating or Adding to a Note File

NOTE.BAT creates a new file or adds to an existing file, time stamping and date stamping each entry. It's a convenient way to keep a collection of dated notes.

Figure 14-5 shows the commands in NOTE.BAT.

```
 1:    @echo off
 2:    if not "%1"=="" goto OK
 3:    echo <Alt-255>
 4:    echo You must enter the name of a file as a parameter.
 5:    echo If it doesn't exist, it is created.
 6:    echo [7mPlease re-enter the command.[m
 7:    goto END
 8: :OK
 9:    if exist %1 goto SHOW_IT
10:    echo ------------------- NOTE FILE %1 > %1
11: :SHOW_IT
12:    cls
13:    more < %1
14:    echo <Alt-255>
15:    echo Type the line or lines to be added to %1.
16:    echo [7mAfter the last line, press {F6} and {ENTER}.[m
17:    echo <Alt-255> >> %1
18:    date < cr.dat ¦ find "C" >> %1
19:    time < cr.dat ¦ find "C" >> %1
20:    echo ------------------------------ >> %1
21:    copy %1+con note.$$$/b > nul
22:    erase %1
23:    rename note.$$$ %1
24: :END
```

Figure 14-5. *NOTE.BAT: Creating or adding to a note file.*

The following explanations describe how NOTE.BAT works:

- Line 2 skips the following error message if you specify the name of a file.
- Lines 3–7 warn you that you must specify the name of a file as a parameter, then return to DOS.

- Lines 9 and 10 create the file if it doesn't exist. (There are 20 dashes in the title line.)
- Lines 11–13 clear the screen and display the file. (If the file is new, it contains just the title line.)
- Lines 14–17 display a blank line, instructions, and another blank line.
- Lines 18–20 write the current date, current time, and a separator line to the file. (There are 30 dashes.)
- Line 21 copies the existing file and any lines added from the console to a temporary file named NOTE.$$$.
- Lines 22 and 23 erase the existing file and change the name of the temporary file from NOTE.$$$ to the file name you specified.

Testing NOTE.BAT

NOTE.BAT requires the file CR.DAT, described in Chapter 6. If you haven't yet created CR.DAT, do so now, and place it in the same directory with NOTE.BAT. To test NOTE.BAT, first enter the command with no parameters:

```
C>note
```

NOTE.BAT displays the following warning and returns to DOS:

```
You must enter the name of a file as a parameter.
If it doesn't exist, it is created.
Please re-enter the command.
```

Enter the command again, specifying NOTETEST.DOC; NOTE.BAT prompts you to enter the lines to be added:

```
C>note notetest.doc
-------------------- NOTE FILE notetest.doc

Type the line or lines to be added to notetest.doc.
After the last line, press {F6} and {ENTER}.

This could be the start
of something big.
^Z
C>_
```

NOTE.BAT saves the file and returns to DOS.

Enter the following to add to NOTETEST.DOC; NOTE.DOC displays the contents of the file and prompts you to enter more lines:

```
C>note notetest.doc

-------------------- NOTE FILE notetest.doc

Current date is Fri 10-16-1992
Current time is 10:45:55.72
------------------------------

This could be the start
of something big.

Type the line or lines to be added to notetest.doc.
After the last line, press {F6} and {ENTER}.

This is the next entry.
^Z
```

If you display the file with the Type command, you'll see the second entry with its time and date stamps.

Creating a Fixed-Format File

MEMO.BAT creates a file in fixed format—a simple memo with to, from, and subject lines. It includes an option for printing the file.

Figure 14-6 shows the commands in MEMO.BAT.

```
 1:    @echo off
 2:    if "%1"=="" goto NO_FILE
 3:    if not exist %1 goto ENTER
 4:    echo [7mThat file exists. If you continue, you'll replace it.[m
 5:    goto QUIT
 6: :NO_FILE
 7:    echo You didn't specify a file name.
 8:    echo You'll only be able to print the memo.
 9: :QUIT
10:    echo [7mPress F1 to continue; any other key quits.[m
11:    reply
12:    if errorlevel 59 if not errorlevel 60 goto ENTER
13:    goto END
14: :ENTER
15:    cls
```

Figure 14-6. *MEMO.BAT: Creating a fixed-format text file.* *(continued)*

Figure 14-6. *continued*

```
16:    echo [7mPress {Enter}{F6}{Enter} after TO.[m
17:    type \batch\memoto.dat
18:    copy \batch\memoto.dat+con memo.$$$ > nul
19:    echo [7mPress {Enter}{F6}{Enter} after FROM.[m
20:    type \batch\memofrom.dat
21:    copy memo.$$$+\batch\memofrom.dat+con > nul
22:    echo [7mPress {Enter}{Enter}{F6}{Enter} after SUBJ.[m
23:    type \batch\memosubj.dat
24:    copy memo.$$$+\batch\memosubj.dat+con > nul
25:    echo [7mType the memo. Press {F6} and {Enter} at the end.[m
26:    copy memo.$$$+con > nul
27:    if not "%1"=="" copy memo.$$$ %1 > nul
28:    cls
29:    type memo.$$$
30:    echo <Alt-255>
31:    echo [7mPress F1 to print the memo, any other key to quit.[m
32:    reply
33:    if errorlevel 59 if not errorlevel 60 copy memo.$$$ prn > nul
34:    erase memo.$$$
35: :END
```

The following explanations describe how MEMO.BAT works:

- Line 2 goes to display a warning message if you didn't specify a file name.
- Line 3 skips the following warning message if you didn't specify an existing file.
- Lines 4 and 5 display a message that warns you that there is a file with the name you specified, and go on to give you the options of continuing or quitting.
- Lines 7 and 8 display a message that warns you that you didn't specify a file name.
- Lines 9–13 display a message telling you that you can press F1 to continue; wait for you to press a key; then return to DOS if you press any key other than F1.
- Lines 15–17 clear the screen, display instructions for entering the recipient's name, and type MEMOTO.DAT (which contains TO:).
- Line 18 copies MEMOTO.DAT plus any lines from the console to the temporary file MEMO.$$$ (which now contains the name of the recipient).

- Lines 19–21 display instructions for entering the sender's name, type MEMOFROM.DAT (which contains FROM:), then copy MEMO.$$$ plus MEMOFROM.DAT plus any lines from the console to MEMO.$$$ (adding the sender's name).
- Lines 22–24 display instructions for entering the subject, type MEMOSUBJ.DAT (which contains the line SUBJ:), then copy MEMO.$$$ plus MEMOSUBJ.DAT plus any lines from the console to MEMO.$$$ (adding the subject).
- Lines 25 and 26 display instructions for entering the body of the memo, then copy MEMO.$$$ plus any lines from the console to MEMO.$$$ (adding the body of the memo).
- Line 27 copies the memo to the file you specified if you specified a file name, and redirects display output to nul.
- Lines 28–30 clear the screen and then display the memo followed by a blank line.
- Lines 31–33 tell you that you can print the memo by pressing F1, wait for you to press a key, and print the memo if you press F1.
- Line 34 erases the temporary file MEMO.$$$.

Testing MEMO.BAT

MEMO.BAT requires three temporary files that contain the fixed parts of the file in the directory \BATCH. Enter the following to create the files. Note that as you create each file you do *not* press Enter until after you press Ctrl-Z, and that there is a space after the colon and before the Ctrl-Z.

```
C>copy con \batch\memoto.dat
TO: ^Z
        1 file(s) copied

C>copy con \batch\memofrom.dat
FROM: ^Z
        1 file(s) copied

C>copy con \batch\memosubj.dat
SUBJ: ^Z
        1 file(s) copied
```

Now test MEMO.BAT by entering the command with no parameters:

```
C>memo
```

MEMO.BAT should display a warning and give you the option of returning to DOS:

```
You didn't specify a file name.
You'll only be able to print the memo.
Press F1 to continue; any other key quits.
```

Press F1. The following dialogue shows how MEMO.BAT works: The reverse-video text prompts you to enter the parts of the memo. The other text displays your responses to create a short memo. Because the use of the Enter key is important to the spacing of the memo, this example tells you when to press Enter:

```
Press {Enter}{F6}{Enter} after TO.
TO: Thomas Watson<Enter>
^Z<Enter>
Press {Enter}{F6}{Enter} after FROM.
FROM: Alexander Graham Bell<Enter>
^Z<Enter>
Press {Enter}{Enter}{F6}{Enter} after SUBJ.
SUBJ: Message<Enter>
<Enter>
^Z<Enter>
Type the memo. Press {F6} and {Enter} at the end.
I must speak with you about<Enter>
your use of the WATS line.<Enter>
<Enter>
Al<Enter>
^Z<Enter>
```

MEMO.BAT clears the screen, displays the memo, and gives you the option of printing it:

```
TO: Thomas Watson
FROM: Alexander Graham Bell
SUBJ: Message

I must speak with you about
your use of the WATS line.

Al
Press F1 to print the memo, any other key to quit.
```

Make sure the printer is on and press F1. The file should be printed.

Because you didn't specify a file name, this memo isn't saved. MEMO.BAT also warns you if you specify the name of an existing file, giving you the option of replacing it or returning to DOS.

You could use this same approach—perhaps even modify MEMO.BAT—to create any sort of fixed-format file that you use, such as electronic mail messages, status reports, or data-entry files.

Adding the Date and Time to a File

STAMP.BAT adds two lines that contain the current date and current time at the beginning of any existing file.

Figure 14-7 shows the commands in STAMP.BAT.

```
 1:    @echo off
 2:    if not "%1"=="" goto OK
 3:    echo [7mYou must specify a file name.[m
 4:    echo Please re-enter the command.
 5:    goto END
 6: :OK
 7:    date < cr.dat ¦ find "C" > stamp.$$$
 8:    time < cr.dat ¦ find "C" >> stamp.$$$
 9:    echo ------------------------------ >> stamp.$$$
10:    copy stamp.$$$+%1 temp.$$$ > nul
11:    erase stamp.$$$
12:    erase %1
13:    rename temp.$$$ %1
14: :END
```

Figure 14-7. *STAMP.BAT: Adding date and time stamps to a file.*

The following explanations describe how STAMP.BAT works:

- Line 2 skips the following error message if you specify the name of a file.
- Lines 3–5 warn you that you must specify a file name as a parameter and return you to DOS.
- Lines 7–9 write a date stamp, time stamp, and dividing line to a temporary file named STAMP.$$$.
- Line 10 copies the temporary file, then the file you named, to a second temporary file named TEMP.$$$.
- Line 11 erases the temporary file named STAMP.$$$.

- Lines 12 and 13 erase the file you named and change the name of the temporary file named TEMP.$$$ to the file name you specified.

Testing STAMP.BAT

STAMP.BAT requires the file CR.DAT, which was described in Chapter 6. If you have not already created CR.DAT, do so now and place it in the same directory as STAMP.BAT. First, enter the command with no parameters:

```
C>stamp
```

STAMP.BAT displays the following warning and returns to DOS:

```
You must specify a file name.
Please re-enter the command.
```

Enter the following to create a test file:

```
C>copy con stamptst.doc
This file should be dated.
^Z
```

Now type the STAMP.BAT command naming STAMPTST.DOC:

```
C>stamp stamptst.doc
```

You'll see some disk activity, then DOS displays the system prompt. Enter the following to see the result:

```
C>type stamptst.doc
Current date is Mon 10-16-1989
Current time is 10:57:23.35
-----------------------------
This file should be dated.

C>_
```

You can use STAMP.BAT to stamp the date and time on any file.

Finding a File on a Hard Drive

It isn't hard to forget where you stored a file, especially if your hard drive contains hundreds of files in a couple of dozen directories. FINDFILE.BAT simplifies the task of finding a file at the expense of a bit of disk space, and usually finds the file in a few seconds.

FINDFILE.BAT takes advantage of the Check Disk (*chkdsk*) command's /v option to display the name of each file. Instead of displaying this list of files, FINDFILE.BAT redirects it to a file named ALLFILES.DAT in the root directory, then searches this file with the Find command for file names that contain the specified string. Because ALLFILES.DAT doesn't require a special directory, you can use FINDFILE.BAT with any disk.

ALLFILES.DAT isn't an especially large file: Its size depends on how many files are on the disk and how long the path names are, but each directory entry averages between 30 and 40 bytes. Estimating generously, then, on a floppy disk with about 100 files ALLFILES.DAT might be 4000 bytes long; on a 40-megabyte hard drive with 2000 files, it might be 90,000 bytes long. This isn't a bad price for the convenience.

Because the Check Disk command can take as long as a minute or more, FINDFILE.BAT doesn't run the command unless you specify *new* as the second parameter or unless ALLFILES.DAT isn't in the root directory. Except when you're doing work that requires changing many files, you won't have to tell FINDFILE.BAT to run the Check Disk command very often, so the search usually takes no more time than it takes DOS to search ALLFILES.DAT.

Figure 14-8 shows the commands in FINDFILE.BAT.

```
 1:    @echo off
 2:    if not "%1"=="" goto OK
 3: :INSTRUCT
 4:    echo <Alt-255>
 5:    echo [1mCommand              Result[m
 6:    echo findfile STRING      Searches for file names that contain STRING.
 7:    echo                      STRING must be entered in uppercase.
 8:    echo findfile STRING new  Forces a Chkdsk command.
 9:    goto END
10: :OK
11:    if "%2"=="new" goto CHKDSK
12:    if not "%2"=="" goto INSTRUCT
13:    if exist \allfiles.dat goto FIND_IT
14: :CHKDSK
15:    echo [7mExecuting Check Disk command.[m
16:    chkdsk /v > \allfiles.dat
17: :FIND_IT
18:    find "%1" \allfiles.dat
19: :END
```

Figure 14-8. *FINDFILE.BAT: Finding a file on a hard drive.*

The following explanations describe how FINDFILE.BAT works:

- Line 2 skips the instructions if you specified at least one parameter.
- Lines 4–9 display instructions for entering the Findfile batch command, and return to DOS.
- Line 11 goes to carry out a Check Disk command if you entered *new* as the second parameter.
- Line 12 goes to display the instructions and return to DOS if you entered anything other than *new* as the second parameter.
- Line 13 goes directly to the search of ALLFILES.DAT if that file exists in the root directory.
- Line 15 displays a message notifying you that a Check Disk command is to be carried out.
- Line 16 carries out a Check Disk command, redirecting the output to ALLFILES.DAT in the root directory. Notice that this command includes both a slash (/v) and a backslash (\allfiles.dat). If you're using DOS 5, you can substitute the command *dir *.*/s* for *chkdsk /v*. That Dir command will produce a similar list of files, and it has the added benefit of being an internal command. The /s switch tells Dir to locate all files in all subdirectories, and since *.* is specified as the file name, that means all files on the disk.
- Line 18 finally searches ALLFILES.DAT for the string you specified.

Testing FINDFILE.BAT

FINDFILE.BAT requires no other files. First enter the command with no parameters:

```
C>findfile
```

FINDFILE.BAT should display the following instructions:

```
Command                 Result
findfile STRING         Searches for file names that contain STRING.
                        STRING must be entered in uppercase.
findfile STRING new     Forces a Chkdsk command.
```

The first time you use it, FINDFILE.BAT runs the Check Disk command because there is no file named ALLFILES.DAT in the root directory. You must enter the search string in uppercase because the output of the Check Disk command is uppercase. Enter the following to display a list of all files whose name contains EXE:

```
C>findfile EXE
```

FINDFILE.BAT responds by telling you that it's running the Check Disk command, then displays the names of the files it found. Because it uses the Find command, the list of files is preceded by a line that identifies the file searched (\ALLFILES.DAT). The output might look like this:

```
Executing Check Disk command.

---------- \ALLFILES.DAT
      C:\AUTOEXEC.BAT
      C:\DOS\ATTRIB.EXE
      C:\DOS\FIND.EXE
      C:\DOS\JOIN.EXE
      C:\DOS\SHARE.EXE
      C:\DOS\SORT.EXE
      C:\DOS\SUBST.EXE
```

Your list will be different, of course, but it should include the above files (although not necessarily in the same order or directory). Notice that in addition to the files whose extension is EXE, the list also includes AUTOEXEC.BAT. (It may be in a different position in the list, depending on where it appears in your root directory.)

You can limit the output to just files with the extension EXE by including the period in the string to find. Enter the Findfile batch command again:

```
C>findfile .EXE
```

This time there's no message, and the command takes much less time because it doesn't have to run the Check Disk command. And there's no AUTOEXEC.BAT in the new list of files. From now on, each search will be this fast until you specify *new* to force another Check Disk command, or erase ALLFILES.DAT from the root directory.

Whenever you want to make a new copy of ALLFILES.DAT, add *new* as a second parameter following the search string.

Quick and Safe Disk Formatting

DISKFORM.BAT provides both safety and convenience when formatting floppy disks. It replaces FORMAT.COM as the command you use to format a disk, and will format only the disk specified in the batch file—never a hard drive. It beeps when it's ready for you to insert a floppy disk, so you don't have to sit watching the screen while you format a new box of disks.

DISKFORM.BAT uses the program REPLY.COM, described in Chapter 7. If you have not yet created REPLY.COM, do so now, and place it in the same directory with DISKFORM.BAT.

Figure 14-9 shows the commands in DISKFORM.BAT.

```
 1:    @echo off
 2:    cls
 3: :NEW_DISK
 4:    echo <Alt-7>Put the diskette to be formatted in drive A.
 5:    echo [7mPress F1 to format, any other key to stop.[m
 6:    echo <Alt-255>
 7:    reply
 8:    if errorlevel 59 if not errorlevel 60 goto OK
 9:    goto END
10: :OK
11:    cls
12:    xformat a: < \batch\formrply.dat
13:    goto NEW_DISK
14: :END
15:    echo <Alt-255>
16:    echo ** Formatting stopped.
```

Figure 14-9. *DISKFORM.BAT: Quick and safe disk formatting.*

The following explanations describe how DISKFORM.BAT works:

- Lines 2–6 clear the screen and display the instructions followed by a blank line. The bell (Alt-7) in line 4 is used because it's central to one of the purposes of DISKFORM.BAT.
- Line 7 waits for you to press a key.
- Line 8 proceeds with formatting if you press F1.
- Line 9 returns to DOS.
- Line 12 executes the Format command (renamed XFORMAT.COM) with input redirected from \BATCH\FORMRPLY.DAT, which contains a carriage return to start formatting, a carriage return to specify no disk label (if you don't have DOS version 4 or later, don't include this), and an N to stop formatting any more disks.
- Line 13 goes back to display the instructions again.
- Lines 15 and 16 display a blank line and tell you that the batch file has ended.

Before testing DISKFORM.BAT, change the name of the FORMAT.COM program to XFORMAT.COM. If you have already renamed FORMAT.COM, change *xformat* in line 12 of Figure 14-8 to the new name.

DOS 5 will cancel the formatting if you try to reformat a disk that was formatted at a different density. If you want to format disks regardless of any previous formatting, use the /u switch after the XFORMAT command in line 12. This will cause an unconditional format to be conducted, but you will not be able to unformat the disk and recover accidentally deleted files. (See below.)

As shown in Figure 14-9, DISKFORM.BAT formats only floppy disks in drive A. If you format disks in both drive A and drive B, you could name the version shown in Figure 14-9 FORMDRVA.BAT, then change *a:* in line 12 of Figure 14-9 to *b:* and save that version as FORMDRVB.BAT. If you don't want two batch files for formatting disks, you could change *a:* in line 12 of Figure 14-9 to *%1:* and type the drive letter with the Diskform batch command. The choice depends on whether you prefer to limit the number of files or to avoid entering a parameter with the command.

Testing DISKFORM.BAT

DISKFORM.BAT requires the file FORMRPLY.DAT in the BATCH directory. Create it by entering:

```
C>copy con \batch\formrply.dat
<Enter>
<Enter>
N<Enter>
^Z<Enter>
```

Have a disk ready that you can format, and enter the command:

```
C>diskform
```

DISKFORM.BAT responds by beeping, clearing the screen, and displaying its instructions:

```
Put the diskette to be formatted in drive A.
Press F1 to format, any other key to stop.
```

Put the disk to be formatted in drive A and press F1. DISKFORM.BAT clears the screen again, and now the DOS Format command displays its instructions and starts formatting the disk:

```
Insert new diskette for drive A:
and strike ENTER when ready...
```

You don't have to press Enter because DISKFORM.BAT redirects the input of the Format command from FORMRPLY.DAT, which contains two carriage returns and the letter N.

When the disk is formatted, the Format command displays its message that shows how much space is available on the disk, then DISKFORM.BAT beeps and displays its instructions again.

This time press the Spacebar (or any other key except F1) to stop DISKFORM.BAT.

Accidents do happen, and someday, although all users dread it, an unintentional format may occur. Pray it's not the hard drive. As luck would have it, starting with version 5.0, DOS includes an unformatting utility. For more information on this amazing *deus ex machina,* refer to Chapter 16, "A Survival Guide for the Nineties."

Restoring the Current Directory

It can be disconcerting, inconvenient, or just a pain if a batch command leaves you in a directory different from the one in which you entered it. But even if a batch command must change the directory to carry out its mission, you can make it restore the directory before it returns to DOS.

The technique requires only four additional commands in a batch file:

1. A Change Directory (*cd*) command whose output (the name of the current directory) is redirected to a temporary file.
2. A Copy command that combines this temporary file with a file that contains *cd* followed by a space, creating a batch file that changes to the directory whose name was saved in step 1.
3. An Erase command that erases the temporary file created in step 1.
4. The batch command created in step 2, which must be in a directory in the command path so that it can be used from any directory.

As usual, it's more complicated in the telling than in the doing. You can test the batch file in a few moments.

The example assumes that you have a directory named \BATCH used for batch files; if you don't, create it or substitute the name of some other directory that you use for batch files wherever you see *\batch* in the example. Make sure the other directory is in your command path.

To follow the example, you need the file that contains *cd* followed by a space. This isn't a temporary file; it must always be available to any batch file that restores the directory. Enter the following to create CD.CMD:

```
C>copy con \batch\cd.cmd
cd ^Z
```

Notice that you don't press Enter after *cd*, but only after you press the Spacebar and Ctrl-Z. That's because you're going to add the temporary file that contains the current directory to this file to create a Change Directory command; if there were a carriage return after *cd*, DOS would end the batch command at that point and simply display the current directory. If you pressed Enter before you pressed Ctrl-Z, simply repeat the Copy command.

Now you're ready to test the commands required to save and restore the current directory:

1. Redirect the output of a Change Directory command to create a temporary file named CD.$$$:

```
C>cd > cd.$$$
```

2. Create the batch file that restores the directory by combining CD.CMD and CD.$$$ with a Copy command:

```
C>copy \batch\cd.cmd+cd.$$$ \batch\restdir.bat > nul
```

3. Erase CD.$$$:

```
C>erase cd.$$$
```

   Change to the root directory (or any other directory on the current disk):

```
C>cd \
```

4. Enter the RESTDIR.BAT command to return to the directory from which you started:

```
C>restdir
```

To make any batch file restore the current directory, put the Change Directory, Copy, and Erase commands anywhere before the batch file changes the directory, and end the batch file with the Restdir batch command.

The Erase command isn't really required, but if you didn't erase CD.$$$, a file with that name would be left in each directory in which you used a batch command that restored the current directory. It isn't necessary to erase RESTDIR.BAT, however, because a new version is created in BATCH (or whatever directory you use for batch files) each time.

Another way to do this is by incorporating three lines into any batch file: two to create a second batch file to save the current directory, and a third that restores that directory. Again, the file CD.CMD is used.

You could place the following two lines of code at the start of a batch file in which you want to save the current directory:

```
copy \batch\cd.cmd %temp%\back2dir.bat
cd >> %temp%\back2dir.bat
```

The first line copies CD.CMD from the BATCH subdirectory. (Remember to substitute whichever directory you keep your CD.CMD file in.) This is copied to a file called BACK2DIR.BAT, located in the temporary subdirectory identified by the environment variable *%temp%*. If you're not using DOS 5, then substitute the name of a temporary subdirectory, or just use the BATCH directory or the root directory.

The next line takes the output of the Change Directory command—the current directory—and tacks it to the end of BACK2DIR.BAT. Once done, the file BACK2DIR.BAT will contain one line, something like the following:

```
cd c:\DOS
```

To return to the original directory, put this Call command at the end of your batch file:

```
call %temp%\back2dir.bat
```

You can also use *back2dir* as the last command, since control may not have to return to the current batch file, or just use it by itself to restore the original subdirectory.

Reducing the Drudgery of Backup

You know that you should back up your hard drive frequently, but do you take the time to do it? Just how often you should back up depends on how much you use your system, of course; a good rule of thumb is to back up right before something bad happens. Since you don't know when that may be, back up whenever losing the files you haven't backed up is too painful to contemplate. But most of us are creatures of habit, so a more effective guideline is to back up at the end of each day that you use your computer.

If you had to back up every file on your hard drive every time you backed up, you probably wouldn't do it every day. You might not even do it every week. Happily, you don't have to back up every file: You've got your original program disks safely tucked away so you can recover them in case something awful happens; and you only have to back up data files—such as word-processing documents, spreadsheets, or graphics images—that have changed since the last time you backed up.

The first time you back up your hard drive, you do have to back up every data file. But after that, you'll only have to back up a few files every day to protect your work.

The DOS Backup and Restore Commands

DOS includes two commands specifically for backing up your hard drive: Backup, which copies files to a floppy disk, tape backup unit, or another hard drive, and Restore, which copies files back to the hard drive.

Both commands include several parameters that let you be as selective as you like when backing up files. You can back up or restore only those files that have changed, or those files that have changed since a particular date, or even a particular time on that date.

The Backup and Restore commands have a couple of drawbacks, however. They don't store the backed-up files as DOS usually stores files, so you can't just copy the files back to the hard drive with the Copy or Xcopy command; you must use the Restore command. And both commands have changed several times as new versions of DOS have been released. As a result, you can't restore files backed up with a different version of DOS.

Hard Drive Backup Programs

Because backing up a hard drive is so important, and because so many people consider backing up such a pain, several backup programs are available that make it easy for you to select the files to be backed up and do the copying much faster than DOS. Some of them also compress the files so that you need fewer floppy disks to hold your backed-up files.

Using one of these programs makes disk backup a fairly simple, painless process. One drawback is their price—ranging from around $50 to more than $150; another is the fact that most of them store the files in a special way, so that you can't simply restore the files with the DOS Copy or Xcopy command but must use the backup program itself to restore the files.

Two Batch Files That Make Backup Easy

The two batch files described here—BAKUP.BAT and DISKBAK.BAT—make up a hard drive backup system that lets you back up with a minimum of time and fuss. After you back up all your data files for the first time, daily backup takes just a few minutes.

The Xcopy command can copy only as many files as will fit on a floppy disk, so the key to using these batch files effectively is to break up your files to be backed up into groups, each backed up on a different disk. You could, for example, back up your word-processing files on one disk, spreadsheet files on another, and miscellaneous files on a third.

You tailor the batch files to your own backup needs by putting commands in BAKUP.BAT that assign a name to each group's backup disk and specify which directories or files are to be backed up. When you run BAKUP.BAT, it calls DISKBAK.BAT once for each group of files to be backed up, prompting you to insert the correct disk and checking to make sure that you have done so.

Within each group, you can back up individual files, groups of files, or directories, and you can limit the backup to the specified directory or all included subdirectories.

You can use either the Xcopy command or the Backup command in DISKBAK.BAT, but Xcopy is a bit easier to work with because it stores files on the backup disks as regular DOS files. Xcopy is shown in the description of DISKBAK.BAT that follows; the section "Backing Up Very Large Files" at the end of the chapter describes when you might need the Backup command and how to change DISKBAK.BAT to use it.

Entering BAKUP.BAT

BAKUP.BAT controls which directories or files are backed up, and calls DISKBAK.BAT to carry out the backup commands. Figure 14-10 shows the commands in BAKUP.BAT. Create the file, but don't type the words in italics in line 5. You'll come back a bit later to tailor BAKUP.BAT to your needs by replacing those lines with Call commands that tell DISKBAK.BAT the name to assign to each backup disk, and to specify which directories or files are to be backed up.

BAKUP.BAT displays some of its messages in reverse video. If you'd rather not use reverse video, omit the ANSI.SYS commands in lines 9 and 10.

```
1:    @echo off
2:    set backup=start
3:    c:
4:    cls
5:    call diskbak Directories to be backed up
6:    if %backup%==cancel goto DONE
      More Call commands, if necessary, to name additional
      directories to be backed up. Each must be followed by
      a duplicate of Line 6.
```

Figure 14-10. *BAKUP.BAT: Defining which files are to be backed up.* *(continued)*

Figure 14-10. *continued*

```
7: :DONE
8:    echo <Alt-255>
9:    if %backup%==cancel echo [7m *** BACKUP CANCELED *** [m
10:   if not %backup%==cancel echo [7m *** BACKUP COMPLETE *** [m
11:   set backup=
```

How BAKUP.BAT works

The following explanations describe how BAKUP.BAT works:

- Line 2 assigns the value *start* to an environment variable named *backup*.
- Lines 3 and 4 change the current drive to C and clear the screen. If you aren't going to back up files from drive C, change the *c* in line 3 to the drive letter of the hard drive you are going to back up.
- Line 5 calls DISKBAK.BAT to back up the specified directory or directories. DISKBAK.BAT requires at least two parameters: The first is displayed as the name of the backup disk to insert, and the second is the path name of the files or directory to be backed up. If the third parameter is /s, all subdirectories in the directory named by the second parameter are backed up. The second and third parameters can be repeated as often as necessary—up to the DOS limit of 127 characters in a command line—to specify additional path names of directories or files to be backed up.
- Line 6 goes to the label DONE to stop backing up files if the value of the environment label *backup* is *cancel*. (This value is set if you press the Escape key in response to the prompt issued by DISKBAK.BAT.)
- Line 8 echoes a blank line.
- Line 9 displays the message *** *BACKUP CANCELED* *** if the value of the environment variable named *backup* is *cancel* (caused by pressing the Escape key in response to the prompt displayed by DISKBAK.BAT).
- Line 10 displays the message *** *BACKUP COMPLETE* *** if the value of the environment variable named *backup* is not *cancel*.
- Line 11 deletes the environment variable named *backup*.

Entering DISKBAK.BAT

DISKBAK.BAT displays messages and prompts, checks your replies, and carries out the Xcopy command that backs up the files. Figure 14-11 shows the commands in DISKBAK.BAT. It's quite a bit longer than BAKUP.BAT

because of the messages and prompts it displays. Create the file. DISKBAK.BAT displays some of its messages in color. If you'd rather not use color, omit the ANSI.SYS commands in lines 3, 5, 9, 20, and 32.

```
1:    @echo off
2:    cls
3:    echo [1;36;44mHard Drive Backup [m
4:    echo <Alt-255>
5:    echo <Alt-7>[1;33mMake sure the [44m %1 [40m
      backup disk is in drive A.
6:    echo <Alt-255>
7:    echo Press F1 to back up
8:    echo       F10 to skip to the next backup disk
9:    echo       ESC to stop backing up and return to DOS[m
10: :GET_RPLY
11:   reply
12:   if errorlevel 69 goto GET_RPLY
13:   if errorlevel 68 goto END
14:   if errorlevel 59 if not errorlevel 60 goto DO_IT
15:   if errorlevel 27 if not errorlevel 28 goto CANCEL
16:   goto GET_RPLY
17: :DO_IT
18:   if exist a:\%1.id goto GOT_DISK
19:   echo <Alt-255>
20:   echo <Alt-7>[7m ** Incorrect %1 backup disk ** [m
21:   echo Press F1 to create ID file; any other key cancels backup.
22:   reply
23:   if errorlevel 59 if not errorlevel 60 goto MAKE_ID
24:   goto END
25: :MAKE_ID
26:   echo %1 ID FILE > a:\%1.id
27: :GOT_DISK
28:   shift
29:   if "%1"=="/s" shift
30:   if "%1"=="" goto END
31:   echo <Alt-255>
32:   echo *** Backing up [31m %1 [m to [31m A:%1 [m
33:   if "%2"=="/s" xcopy c:%1 a:%1 /s /m
34:   if not "%2"=="/s" xcopy c:%1 a:%1 /m
35:   goto GOT_DISK
36: :CANCEL
37:   set backup=cancel
```

Figure 14-11. *DISKBAK.BAT.*

How DISKBAK.BAT works

The following explanations describe how DISKBAK.BAT works.

- Lines 2–4 clear the screen, then display a title message and a blank line.
- Line 5 beeps and displays a prompt advising you to make sure that the correct backup disk (named in the first parameter of the Diskbak command in Line 5 of BAKUP.BAT) is in drive A. If you don't want the beep, leave out the Alt-7.
- Lines 6–9 display a blank line and a menu that tells you that you can proceed with the backup by pressing F1, skip to the next backup disk by pressing F10, or cancel the backup operation by pressing Escape.
- Line 11 waits for you to press a key.
- Line 12 goes back to continue waiting if you press any key whose key code is greater than 68 (the key code of F10).
- Line 13 skips all further backups on the current backup disk by going to the label END (the end of DISKBAK.BAT) if you press F10 (key code 68). This returns control to the command immediately following the Call command in BAKUP.BAT that carried out DISKBAK.BAT.
- Line 14 goes to the label DO_IT, where the files are backed up, if you press F1.
- Line 15 goes to the label CANCEL, where the value *cancel* is assigned to the environment variable named *backup* before ending DISKBAK.BAT, if you press the Escape key.
- Line 16 goes back to continue waiting if you press any other key.
- Line 18 goes to the label GOT_DISK to proceed with the backup if there is a file in the root directory of the backup disk whose name matches the first parameter specified with the Call command in BAKUP.BAT (with the extension ID). This file identifies the proper backup disk.
- Lines 19–26 are carried out only if the proper ID file isn't in the root directory of the backup disk:
  - Lines 19–20 echo a blank line, beep, and display a message warning you that the incorrect backup disk is in drive A. If you don't want the beep, leave out the Alt-7.
  - Lines 21 and 22 prompt you to press F1 to create the ID file or any other key to cancel the backup, then wait for you to press a key.
  - Line 23 goes to the label MAKE_ID to create the ID file if you press F1.
  - Line 24 skips all further backups on the current backup disk by going to the label END (the end of DISKBAK.BAT) if you didn't press

function key F10 (key code 68). This returns control to the command immediately following the Call command in BAKUP.BAT that carried out DISKBAK.BAT.

  - Line 26 creates the ID file by redirecting the output of an Echo command to a file whose name matches the first parameter specified with the Call command in BAKUP.BAT and whose extension is ID.

- Line 28 shifts the list of parameters one position to the left. The first time this command is carried out, it deletes the first parameter specified with the Call command in BAKUP.BAT (the name of the backup disk). If BAKUP.BAT specifies more than one path name with the Call command, the Shift command will be carried out for the second and succeeding path names.

- Line 29 shifts the list of parameters one position to the left if the next command-line parameter is /s, bringing the next path name to the beginning of the list.

- Line 30 goes to the label END (the end of DISKBAT.BAT) if the first parameter is null, which means there are no more files to back up.

- Lines 31 and 32 display a blank line and a message that tells you which file or directory is being backed up.

- Line 33 carries out an Xcopy command with the /s and /m parameters if the parameter following the path name (specified with the Call command in BAKUP.BAT) is /s. /s backs up the specified file or directory and all included subdirectories; /m backs up only those files whose archive attribute is on, then turns their archive attribute off. Note the *c:* before the first *%1*; if you're going to back up files from a hard drive with a different drive letter, change this *c* to the drive letter of the hard drive.

- Line 34 carries out an Xcopy command with just the /m parameter if the parameter following the path name (specified with the Call command in BAKUP.BAT) is not /s. This backs up only files in the specified directory whose archive attribute is on, then turns off their archive attribute. Note the *c:* before the first *%1*; if you're going to back up files from a hard drive with a different drive letter, change this *c* to the drive letter of the hard drive.

- Line 35 goes back to shift the list of parameters and checks them again.

- Line 37 assigns the value *cancel* to the environment variable named *backup* to tell BAKUP.BAT that you pressed the Escape key to cancel further backup.

Tailoring the Diskbak Commands in BAKUP.BAT

It's time to decide how you're going to back up your files. Think about how you use your computer, and divide your data files into groups that match the way you use your computer. Try to keep each group small enough so that, on any given day, no more than 15 or 20 files will have to be backed up (although this number could be higher if the files are small).

If you have set up your file system with separate directories for different types of work—for example, several subdirectories for various word-processing documents—each group could consist of one or more complete directories; you would use a directory path name to specify the files to be backed up. On the other hand, if you tend to store many files in individual directories and use file names to distinguish among different types of work, each group could consist of all files with a particular file extension or a pattern of characters in the file name. In this case, you would use a path name plus a file name with wildcard characters to specify the files to be backed up.

You'll need a backup disk for each group of files. For each backup disk you'll be using, you must enter a Call command in BAKUP.BAT that calls DISKBAK.BAT, specifying the name of the backup disk and which directories or files are to be backed up. Each Call command must be followed by the line *if %backup%==cancel goto DONE* so that you can cancel the backup process by pressing Escape. The Call and If commands replace lines 5 and 6 of Figure 14-9.

The Diskbak command that follows *call* in BAKUP.BAT has three parameters:

diskbak *diskname pathname* /s

diskname specifies the name of the backup disk. DISKBAK.BAT uses this name when it prompts you to insert the backup disk, and when it checks for the identifying file in the root directory of the backup disk. *diskname* must be a single word (no blanks) and a maximum of eight characters long, because it is used as the name of the identifying file in the root directory of the backup disk.

pathname is the path name of the directory to be backed up. If you don't specify a file name, all files in the directory are backed up. You can add a file name to *pathname* to back up a specific file, or a file name with wildcard characters to back up a group of files with similar names or extensions. You can specify more than one *pathname,* separated by blanks.

To Back Up These Files and Directories	*Enter These Lines in BAKUP.BAT in Place of Lines 5 and 6 of Figure 14-10*
\WP\DOCS to a disk named WP	*call diskbak WP \wp\docs* *if %backup%==cancel goto DONE*
\WP\DOCS*.DOC to a disk named WP	*call diskbak WP \wp\docs*.doc* *if %backup%==cancel goto DONE*
\WP\DOCS*.DOC and all its subdirectories to a disk named WP	*call diskbak WP \wp\docs*.doc /s* *if %backup%==cancel goto DONE*
\WP\DOCS and all its subdirectories to a disk named WP, and \123*.WK1 and all its subdirectories to a disk named 123	*call diskbak WP \wp\docs /s* *if %backup%==cancel goto DONE* *call diskbak 123 \123*.wk1 /s* *if %backup%==cancel goto DONE*
\WP\DOCS and all its subdirectories to a disk named WP; \123*.WK1 and all its subdirectories to a disk named 123; and \PGM and all its subdirectories, \BATCH, and \MAX and all its subdirectories to a disk named MISC	*call diskbak WP \wp\docs /s* *if %backup%==cancel goto DONE* *call diskbak 123 \123*.wk1 /s* *if %backup%==cancel goto DONE* *call diskbak MISC \pgm /s \batch \max /s* *if %backup%==cancel goto DONE*

Figure 14-12. *Entering path names in BAKUP.BAT.*

/s tells DISKBAK.BAT to back up files not only from the immediately preceding directory, but also from all the subdirectories it contains.

Figure 14-12 shows several examples of Call commands to back up different combinations of files and directories. Decide how many backup disks you need, then replace lines 5 and 6 of BAKUP.BAT with the Call commands to name each disk and identify the files to be backed up.

Using the Backup Batch Files

After you have decided how to group your backup files and have added the required Call commands to BAKUP.BAT, it's time to back up all your data files. This could take half an hour or so and require up to a dozen disks, depending on how many files you decide to back up and the capacity of your disks, but remember: You'll only do this once. You must use formatted disks, so format several before you begin. Label one with the name of the first group you specified with the Call command in BAKUP.BAT, and put it in drive A.

Now enter the Backup command. DISKBAK.BAT should respond by displaying a title message, reminding you to make sure the correct backup disk is in drive A, then prompting you to press F1 to proceed with the backup, F10 to skip to the next backup disk, or ESC to cancel the whole

backup procedure. (*name* in the reminder is replaced by the name you assigned to the backup disk for the first group of files to be backed up.)

```
Hard Drive Backup
Make sure the name backup disk is in drive A.

Press F1 to back up
      F10 to skip to the next backup disk
      ESC to stop backing up and return to DOS
```

Press Escape. BAKUP.BAT should respond *** *BACKUP CANCELED* *** and return to the DOS system prompt.

Enter *bakup* again; DISKBAK.BAT should display the same messages and prompts as before. Press F10; DISKBAK.BAT should repeat the display, but this time it should name the backup disk for the second group of files to be backed up. Continue pressing F10 until you have skipped all the backup disks; after you skip the last disk, BAKUP.BAT should respond *** *BACKUP COMPLETE* *** and return to the DOS system prompt.

This time you'll go through with it. Enter *bakup* and, in response to the prompt, press F1 to proceed with the backup. Because there are no files on the backup disk, DISKBAK.BAT can't find the identifying file; it warns you that you have inserted the incorrect backup disk and asks you if you want to create the identifying file. (Again, *name* in the first line is replaced by the name you assigned to the backup disk for the first group of files to be backed up.)

```
** Incorrect name backup disk **
Press F1 to create ID file; any other key cancels backup.
```

You've already seen how to cancel the backup, so press F1 to create the identifying file and continue with the backup. DISKBAK.BAT creates the identifying file on the backup disk, displays the file source and destination parameters, and tells you that the Xcopy command has started reading the source files:

```
Reading source file(s)...
```

This is followed by a list of the files that the Xcopy command reads into the computer's memory and writes onto the backup disk. If DOS displays the message *Does filename specify a file name or directory name on the target?*, just press D to tell DOS that you mean a directory.

If there are more files to be copied, the Xcopy command displays *Reading source file(s)...* again, then writes the next batch of files onto the backup disk. This sequence continues until either all the files in this group have been backed up or the backup disk fills.

When a Backup Disk Fills

When the Xcopy command finds that there isn't enough room on a disk to store a file, it stops copying files and displays *Insufficient disk space* followed by a message that tells you the number of files it copied before it ran out of space. This can happen when you first back up all your data files or during a daily backup after you have backed up many files.

There's no problem here. Just write *#1* on the disk label, then label another blank, formatted disk with the name of the group of files and *#2*. Put the disk in drive A and type *bakup*; because the Xcopy command in DISKBAK.BAT backs up only files whose archive attribute is on and then turns off the archive attribute after it copies the file, the backup process resumes with the file that wouldn't fit on the backup disk and continues as before.

When all the files in the first group have been backed up, DISKBAK.BAT displays its opening message and prompt again, this time naming the second group of files you specified with the Call commands in BAKUP.BAT. Proceed as you did with the first group, labeling and using additional backup disks as needed. Continue until all groups of files have been backed up and store the backup disks in a safe place. You now have a copy of each data file that you want to protect on your hard drive.

Now You Can Back Up Every Day

Daily backups are a snap with these batch files. Just type *bakup* and put in the backup disks when DISKBAK.BAT prompts you for them. If you know that you didn't change any files in a particular group, just press F10 to skip that disk. If a backup disk fills, label another and continue. It shouldn't take more than a couple of minutes to back up all your work.

Now, if something happens to a file on your hard drive, simply put the backup disk in drive A and copy the file back with the Xcopy or Copy command. Even if you have to check through several backup disks to find the backup copy, you're still protected against losing the file.

Backing Up Very Large Files

Some programs, especially database programs, can create files larger than the capacity of a floppy disk. The Xcopy command can't copy a file this large, so you must use the Backup command instead of the Xcopy command in DISKBAK.BAT.

In addition to replacing *xcopy* with *backup* in lines 33 and 34 of DISKBAK.BAT, you should also add the /a parameter to the /m and /s parameters in the Backup command so that it will add your daily backup files to the backup disk; if you don't specify /a, the Backup command erases all files on the backup disk before copying the files to be backed up, and all that your backup disk would contain is the most recent daily backup files.

And remember, you must use the Restore command to restore files that are backed up with the Backup command. In general, the Xcopy command works much better in these batch files; use the Backup command only if you'll be backing up files larger than the capacity of your backup disks.

The batch files in this chapter give you several useful ways to work with files, and may give you ideas for some others. Batch files are like tools: They accumulate in cabinets and drawers, but it sure is handy to have the right one around when you need it.

CHAPTER

15

Putting It All Together: Your Own Menu System

It's not all that unusual, unfortunately: You've got an hour or so to edit and print a file; enter some data into a spreadsheet, recalculate, and print it; send an electronic mail message to your project team; and make a quick outline of your presentation at this afternoon's staff meeting.

Here's a not-so-usual solution:

- Press two keys and your word processor is running in the directory that contains the document you need.
- Quit the word processor, press two more keys, and your spreadsheet program is running in the directory that contains the spreadsheet file you need.
- Quit the spreadsheet, press another couple of keys, and the communications program is running.
- Quit the communications program, press a key, and your outline processor is running.
- You finish the outline, print it, and press one last key. As you head for the door, the screen goes blank to protect the phosphor.

You didn't type any commands to change the directory or start a program; you just used the function keys to select options from a series of menus tailored to the programs and directories you use. That's what this chapter is all about—in fact, that's what this book is about: giving you control so that your system takes care of as much housekeeping as possible and you can concentrate on getting your work done.

The Menu System

This chapter shows you how to create a menu system that lets you run any application program, in any directory, by pressing one or two keys. The menu system requires some investment of your time, but the payoff is worth it: For most routine work using application programs, you may never again need to type a command to change the directory or to start an application program.

The menu system includes a Main menu with sub-menus, and incorporates the Printer Setup menu described in Chapter 12. The Main menu includes two types of menu selections:

- A program selection changes the directory and starts a program that uses only one directory.
- A sub-menu displays a menu of several directories that contain files you use with the same program, such as a word processor or a spreadsheet.

You use the same key (F10) to return from a sub-menu to the Main menu and to return from the Main menu to DOS. The F10 key always takes you back one level in the menu structure.

This chapter shows you how to enter and test the Main menu, how to incorporate the Printer Setup menu, and how to add your own program selections and sub-menus. It also shows you how to use the menu system with a RAM disk, and even how to customize AUTOEXEC.BAT so that the menu system runs on a RAM disk, if one is defined, or on a real disk.

Appendix F shows the contents of each file required for the menu system described in this chapter.

Preparing for the Menu System

The menu system needs the programs REPLY.COM, NOCURS.COM, and NORMCURS.COM. If you haven't created REPLY.COM, turn to "Making Batch Commands Interactive: REPLY.COM" in Chapter 7 and follow the instructions listed there. If you haven't created both NOCURS.COM and NORMCURS.COM, turn to "Don't Curse the Cursor" in Chapter 11.

Before you start working on the menu system, type the following commands to create the MENU directory and change to that directory:

```
C>md \menu

C>cd \menu
```

Use the Path command to add \BATCH to your command path so that MAINMENU.BAT can find the programs it needs. If you entered the Printer Setup menu described in Chapter 12, copy all four files (PRNTMENU.BAT, PRNTMENU.DOC, PRNTOPT1.DOC, and PRNTOPT2.DOC) to the MENU directory.

Entering the Menu System

The menu system requires a minimum of two new files: the first is MAINMENU.DOC and the second is MAINMENU.BAT. You'll start by entering a skeleton Main menu, then gradually add program selections and sub-menus. Each sub-menu requires one additional file, which you create by copying and modifying the file MAINMENU.DOC.

The text file: MAINMENU.DOC

MAINMENU.DOC contains the Main menu as it appears on the screen. It contains text and ANSI.SYS commands to control display attributes, but no

DOS commands. You can create it with any text editor or word processor that accepts characters in the extended character set and that lets you save a file without formatting codes.

Figure 15-1 shows the contents of MAINMENU.DOC.

If you entered the Printer Setup menu in Chapter 12, you've already done most of the work. Copy PRNTMENU.DOC to MAINMENU.DOC and revise the copy to match Figure 15-1.

If you didn't create the Printer Setup menu, you'll need it in a moment, so you might as well enter it now. Turn to "Entering PRNTMENU.DOC" in Chapter 12 and follow the instructions there. The instructions include some hints for using your word processor's cut-and-paste capabilities to simplify typing the file. When you have finished entering the file, save it as PRNTMENU.DOC, then revise it to match Figure 15-1 and save it as MAINMENU.DOC.

If you're using a RAM disk, copy MAINMENU.DOC (and if you just entered it, PRNTMENU.DOC) to the MENU directory on a real disk so you won't lose any work if something goes wrong.

Check your menu by displaying MAINMENU.DOC with the Type command.

```
1:
2:  [1m  F O R S Y T H - W O L F  C O M M U N I C A T I O N S  [m
3:
4:
5:                [1mWriting and Business Management System[m
6:
7:
8:
9:
10:  F1  Blank the screen                F2  Selection or sub-menu
11:
12:
13:  F3  Selection or sub-menu           F4  Selection or sub-menu
14:
15:
16:  F5  Selection or sub-menu           F6  Selection or sub-menu
17:
18:
19:  F7  Selection or sub-menu           F8  Selection or sub-menu
20:
21:
22:  F9  Selection or sub-menu           F10 Return to DOS
23:
```

Figure 15-1. *MAINMENU.DOC: Skeletal version of the Main menu.*

```
C>type mainmenu.doc
```

If something isn't right, correct the file and display it again. Then copy the final version to an actual disk if you're using a RAM disk.

The batch file: MAINMENU.BAT

The menu-system batch file has a simple structure. It displays the file MAINMENU.DOC and waits for you to press a key. If you press any key from F1 through F10, MAINMENU.BAT carries out the commands required by that menu selection; it ignores any other key.

Because MAINMENU.BAT gets pretty long by the time you define all 10 selections, you'll start by entering a skeleton that includes only two selections: F1 to blank the screen and F10 to return to DOS. Guided by examples, you'll then add your program selections and sub-menus. Figure 15-2 shows the commands in MAINMENU.BAT.

Use the cut-and-paste capabilities of your text editor or word processor to simplify MAINMENU.BAT:

1. Enter lines 1 through 9.
2. Copy line 9 nine times.
3. Change lines 10 through 18 to match Figure 15-2.
4. Enter lines 19 through 24.
5. Copy lines 23 and 24 seven times.
6. Change the labels in lines 25 through 37 to match Figure 15-2.
7. Enter lines 39 through 45.

Save the MAINMENU.BAT file. If you're using a RAM disk, be sure to copy MAINMENU.BAT to the directory MENU on a real disk.

```
 1:    @echo off
 2: :START
 3:    cls
 4:    cd \menu
 5:    type mainmenu.doc
 6: :GET_RPLY
 7:    reply
 8:    if errorlevel 69 goto GET_RPLY
 9:    if errorlevel 68 goto F10
10:    if errorlevel 67 goto F9
11:    if errorlevel 66 goto F8
```

Figure 15-2. *MAINMENU.BAT: Skeleton of the menu-system batch file.* *(continued)*

Figure 15-2. *continued*

```
12:    if errorlevel 65 goto F7
13:    if errorlevel 64 goto F6
14:    if errorlevel 63 goto F5
15:    if errorlevel 62 goto F4
16:    if errorlevel 61 goto F3
17:    if errorlevel 60 goto F2
18:    if errorlevel 59 goto F1
19:    goto GET_RPLY
20: :F10
21:    cls
22:    goto END
23: :F9
24:    goto GET_RPLY
25: :F8
26:    goto GET_RPLY
27: :F7
28:    goto GET_RPLY
29: :F6
30:    goto GET_RPLY
31: :F5
32:    goto GET_RPLY
33: :F4
34:    goto GET_RPLY
35: :F3
36:    goto GET_RPLY
37: :F2
38:    goto GET_RPLY
39: :F1
40:    cls
41:    nocurs
42:    reply
43:    normcurs
44:    goto START
45: :END
```

How MAINMENU.BAT works

As you might guess from the patterns of commands in MAINMENU.BAT, its operation is fairly simple. It changes the current directory to the MENU directory, displays the Main menu, then waits for you to press a key. If you press a function key, it goes to the corresponding label to carry out the selection; if you press any other key, it goes back to GET_RPLY to wait for you to press another key. (In this skeleton form, it also goes back to GET_RPLY if you press F2 through F9, but you'll soon change that.)

Here's how MAINMENU.BAT works.

- Lines 1–5 clear the screen, change to the MENU directory, and display the Main menu.
- Line 7 waits for you to press a key.
- Line 8 goes back to wait for you to press another key if the code of the key you press is 69 or greater (the highest code you're looking for is 68, which is F10).
- Lines 9–18 go to the label that corresponds to the function-key label (F10 through F1) if you press a function key.
- Line 19 waits for you to press another key if the code of the key you press is less than 59 (the lowest code you're looking for is 59, which is F1).
- Lines 21 and 22 clear the screen and return to DOS if you press F10.
- Lines 23–38 go back to wait for you to press another key if you press F9 through F2. The *goto GET_RPLY* commands in these lines are temporary; you'll replace them as you add program selections and sub-menus to the Main menu.
- Lines 40 and 41 blank the screen if you press F1 by clearing the screen and making the cursor invisible. Line 42 waits for you to press a key. No matter what key you press, line 43 restores the cursor and line 44 goes back to display the Main menu.

Although MAINMENU.BAT will grow as you add program selections and sub-menus, this basic structure will remain unchanged. The labels are descriptive enough to remain accurate landmarks even when you have completed all 10 choices.

Testing MAINMENU.BAT

If the current directory contains MAINMENU.DOC, MAINMENU.BAT, NOCURS.COM, NORMCURS.COM, and REPLY.COM—and if you're using a RAM disk, you have copied MAINMENU.DOC and MAINMENU.BAT to a real disk—you're ready to test the Main menu. Enter the following menu-system command:

```
C>mainmenu
```

The screen should clear and then display the Main menu. If it does, skip the next section and go on to "Testing the Main menu."

What if it doesn't work?

Troubleshooting your creation can be frustrating, so use the clues that DOS gives you. If a menu system doesn't behave as it should, watch for error messages that DOS may display.

- *Bad command or file name* could mean that MAINMENU.BAT or some other command or batch file isn't in the current directory (or a directory that is part of your command path), or that you misspelled its name either when you created it or when you typed a command. Some possible remedies:
  - Make sure that you spelled *mainmenu* correctly when you typed the command to start the menu system.
  - Make sure \BATCH is in the command path.
  - Make sure that the current directory is set to the MENU directory (or the directory you're using for your menu system, if it's different).
  - Display the directory to make sure that the required batch files are there and that their names are spelled correctly.
  - If a batch command is used in another batch file, make sure the name of the batch command is spelled correctly. If the batch command isn't the last command in the batch file, make sure the Call command is used (for example, the line *call prntmenu mainmenu* in MAINMENU.BAT).
- *File not found* could mean that MAINMENU.DOC or some other text file isn't in the current directory, or that you misspelled its name either when you created it or when you entered it in a batch file. Some possible remedies:
  - Make sure that the current directory is set to the MENU directory (or the directory you're using for your menu system, if it's different).
  - Display the directory to make sure that the required text files are there and that their names are spelled correctly.
  - Check the Type command in the batch file that displays the text file (for example, *type mainmenu.doc* in MAINMENU.BAT) to make sure that the name of the text file is spelled correctly.

If you make sure this skeleton form of the menu system works correctly, then add program selections and sub-menus one at a time, testing each as you add it, you shouldn't have to do much troubleshooting.

Testing the Main menu

For the rest of this chapter, if the menu system doesn't react as it should and the reason isn't readily apparent, go back to the heading "What if it doesn't work?" for suggestions about solving the problem.

Display the Main menu, then press F1. After a short delay, the screen should go blank. This takes a moment because MAINMENU.BAT checks to see which key you pressed in high-to-low order (see lines 9–18 of Figure 15-2); F1 is the last key it checks for. You'll find the higher the function-key number you press, the more quickly the menu responds.

Press any key; the screen should display the Main menu again.

Now press F2 through F9. None of those keys should have any effect, other than to cause MAINMENU.BAT to run the REPLY.COM program from disk again. When you press F10, the screen should clear and DOS should display the system prompt.

Adding a program selection to the Main menu

Adding a program selection to the Main menu requires few changes. This section describes the general technique, then gives an example. Use the example as a guide when adding your programs to the Main menu.

To add a program selection to the Main menu:

- Make the program name a Main menu selection by changing the appropriate line in MAINMENU.DOC.
- Insert two commands at the appropriate label in MAINMENU.BAT to change the directory and start the program.
- Find the line in MAINMENU.BAT that follows the two commands you inserted. Change *goto GET_RPLY* to *goto START*.

Figure 15-3 shows the steps required to make F8 in the Main menu run MaxThink (an outline processor); it also shows how the affected portion of MAINMENU.BAT would look after the addition.

To add your own program, choose a function key and a program you use with a single directory. (You'll add a sub-menu a bit later to handle programs that you use with more than one directory.) Using Figure 15-3 as a guide, change the function-key legend in MAINMENU.DOC and insert the new commands in MAINMENU.BAT.

Fire up the Main menu by entering *mainmenu*, and press the function key you chose; the program you selected should start. End the program, and the Main menu should reappear on the screen. Now press F10 to return to DOS. If something went awry, go back to the section "What if it doesn't work?" for help.

1. Change the legend for F8 (the second occurrence of *Selection or sub-menu* in line 19 of MAINMENU.DOC) to *MaxThink.*
2. Insert the following two lines after the label :F8 in MAINMENU.BAT (line 25 of Figure 15-2):

```
cd \max
max
```

3. Change the command *goto GET_RPLY* after the label :F8 in MAINMENU.BAT (line 26 of Figure 15 -2) to *goto START.*

   The affected portion of MAINMENU.BAT is shown below; the numbered lines are the existing lines in MAINMENU.BAT shown in Figure 15-2:

```
23: :F9
24:     goto GET_RPLY
25: :F8
        cd \max
        max
26:     goto START
27: :F7
```

Figure 15-3. *Adding a program selection to the Main menu.*

Incorporating the Printer Setup Menu into the Menu System

If you haven't yet entered the Printer Setup menu and want to include it in the menu system, turn to "The Printer Setup Menu" in Chapter 12. Follow the instructions there, then continue with this section. If you don't want to include the Printer Setup menu, skip this section and turn to "Adding a Sub-Menu to the Main Menu."

To incorporate the Printer Setup menu, copy the files PRNTMENU.BAT, PRNTMENU.DOC, PRNTOPT1.DOC, and PRNTOPT2.DOC to the MENU directory (if they aren't there already). Make the following changes to MAINMENU.DOC and MAINMENU.BAT:

1. Change the legend for F9 in MAINMENU.DOC (the first occurrence of *Selection or sub-menu* in line 22 of Figure 15-1) to *Printer Setup menu.* This is the only change to MAINMENU.DOC.
2. Insert the following command after the label :F9 of MAINMENU.BAT (line 23 of Figure 15-2).

```
call prntmenu mainmenu
```

3. Change the command *goto GET_RPLY* after the label :F9 in MAINMENU.BAT (line 24 of Figure 15-2) to *goto START*.

4. Insert the following command after the line *type prntmenu.doc* in PRNTMENU.BAT (line 3 of Figure 12-18):

   ```
   if "%1"=="mainmenu" echo [22;24Main menu
   ```

   This command changes the F10 legend to read *Return to Main menu* instead of *Return to DOS* if PRNTMENU.BAT is called by MAINMENU.BAT.

The Printer Setup menu should work as it did before; if you haven't used it yet, see "The Printer Setup menu" in Chapter 12. To use the Printer Setup menu from the Main menu, enter:

```
C>mainmenu
```

When the Main menu is displayed, press F9. The Printer Setup menu should be displayed, and the legend for F10 should read *Return to Main menu*. Try some of the selections—F9 to reset the printer. This will tell you right away whether the menu is working. Now press F10 to return to the Main menu, and press F10 again to return to DOS.

If you're using 3.2 or an earlier version of DOS—and, therefore, must use the Command command to call another batch file—you may notice your system prompt flash briefly after the Main menu clears and before the Printer Setup menu is displayed. You can minimize this distraction by adding two commands to MAINMENU.BAT, but it's only an aesthetic problem. If it doesn't bother you, go on to the heading "Adding a Sub-Menu to the Main Menu."

You can't avoid the system prompt, but you can make it shorter. Insert the following two commands in MAINMENU.BAT:

- *prompt* (with no parameter) immediately before *call prntmenu mainmenu* (the Call command you just entered)
- *promptrs* as the last line (after the label :END)

These two commands define the default system prompt (C>) before displaying the Printer Setup menu, then restore your system prompt before returning to DOS. If you haven't yet created a batch file named PROMPTRS.BAT that restores your system prompt (as described in Chapter 8), do it now. (You could simply make the last command of the MAINMENU.BAT file a Prompt command that restores your default

system prompt, but there may be other places where you'll have to restore your system prompt, and it's simpler to use the same batch file everywhere.)

Adding a Sub-Menu to the Main Menu

If you frequently use a word processor or spreadsheet program, you may keep the program itself in one directory and store the document or spreadsheet files in several directories that identify a particular project, department, or client. A sub-menu displays all the directories that you use with one program; no matter what you choose from the sub-menu, the same program is started. The choice determines only which directory becomes the current directory.

Adding a sub-menu involves a bit more work than adding a program selection, but sub-menus significantly expand the usefulness of your menu system. This section describes the general technique, then shows an example; use the example as a guide when adding your sub-menus to the menu system.

To add a sub-menu to the menu system:

- Create a text file that contains the sub-menu. This is fairly simple: Copy MAINMENU.DOC to the new text file, then change the title and menu selections. If the sub-menu has fewer than 10 selections, delete the function-key symbols that aren't required, but leave blank lines in their place, so that the F10 symbol—which returns you to the Main menu—always appears in the same place on the screen. Save the file under the new sub-menu name.
- Put the sub-menu name in the Main menu by changing the legend of the associated function key in MAINMENU.DOC.
- Delete the *goto GET_REPLY* command that follows the label of the associated function key in MAINMENU.BAT.
- Insert the commands that display and manage the sub-menu following the label of the associated function key in MAINMENU.BAT. These commands are a miniature version of MAINMENU.BAT that:
  - Display the sub-menu with a Type command.
  - Wait for a response with REPLY.COM.
  - Check the response with a series of *if errorlevel* commands. If a valid key is pressed, change the directory. If any other key is pressed, go back to wait for a response.
  - Start the application program.
  - Redisplay the sub-menu when the program ends.

To illustrate, suppose you use Microsoft Word. You store document files in four directories (WORD\LETTERS, WORD\MEMOS, WORD\REPORTS, and WORD\PROPOSAL) and you want F3 on the Main menu to display a sub-menu called Word-Processing Directories.

First you would create WORDMENU.DOC, the text file that contains the menu, shown in Figure 15-4. Create it by copying MAINMENU.DOC to WORDMENU.DOC, then change WORDMENU.DOC to match Figure 15-4. Notice that both files contain the same number of lines. The symbols for F5 through F9 are deleted, but the blank lines remain.

```
1:
2:  E[1m F O R S Y T H - W O L F  C O M M U N I C A T I O N S E[m||
3:
4:
5:                    E[1mWord Processing Directories                E[m||
6:
7:
8:
9:
10:  F1  Letters                    F2  Memos
11:
12:
13:  F3  Reports                    F4  Proposal
14:
15:
16:
17:
18:
19:
20:
21:
22:                                 F10  Return to Main Menu
23:
```

Figure 15-4. *WORDMENU.DOC: The word-processing sub-menu.*

Next you would modify MAINMENU.BAT to make F3 display and manage the word-processing sub-menu:

1. Change the legend for F3 (the first occurrence of *Selection or sub-menu* in line 13 of Figure 15-1) to *Word-Processing Directories.*
2. Change the command *goto GET_RPLY* after the label :F3 (line 36 of Figure 15-2) to *Goto F3*:
3. Insert the following commands after the label :F3 (line 35 of Figure 15-2) and before the Goto F3 command you just entered.

```
 1:     cls
 2:     type wordmenu.doc
 3:  :RPLY_WRD
 4:     reply
 5:     if errorlevel 69 goto RPLY_WRD
 6:     if errorlevel 68 goto START
 7:     if errorlevel 63 if not errorlevel 68 goto RPLY_WRD
 8:     if errorlevel 62 if not errorlevel 63 cd \word\proposal
 9:     if errorlevel 61 if not errorlevel 62 cd \word\reports
10:     if errorlevel 60 if not errorlevel 61 cd \word\memos
11:     if errorlevel 59 if not errorlevel 60 cd \word\letters
12:     if errorlevel 0 if not errorlevel 59 goto RPLY_WRD
13:     word
```

- Lines 1 and 2 clear the screen and display the Word-Processing menu.
- Line 4 waits for you to press a key.
- Lines 5 through 12 check which key you pressed and take the appropriate action:
  - Line 5 goes back to wait for you to press another key if the code of the key you pressed was 69 or greater (the highest code you're looking for is 68, which is F10).
  - Line 6 displays the Main menu if you press F10 (code 68).
  - Line 7 goes back to RPLY_WRD to wait for you to press a key if you press F5 (code 63) through F9 (code 67), because only F1 through F4 are valid choices.
  - Lines 8 through 11 change to the directory that corresponds to the selections shown in Figure 15-4 if you press F4 (code 62) through F1 (code 59).
  - Line 12 goes back to RPLY_WRD if the code of the key you press is less than 59.
- Line 13 starts Microsoft Word.

Figure 15-5 shows the affected portion of MAINMENU.BAT after adding the word-processing sub-menu; the numbered lines are existing lines in MAINMENU.BAT as shown in Figure 15-2.

```
33: :F4
34:    goto GET_RPLY
35: :F3
       cls
       type wordmenu.doc
    :RPLY_WRD
       reply
```

Figure 15-5. *MAINMENU.BAT after adding the word-processing sub-menu.* *(continued)*

Figure 15-5. *continued*

```
        if errorlevel 69 goto RPLY_WRD
        if errorlevel 68 goto START
        if errorlevel 63 if not errorlevel 68 goto RPLY_WRD
        if errorlevel 62 if not errorlevel 63 cd \word\proposal
        if errorlevel 61 if not errorlevel 62 cd \word\reports
        if errorlevel 60 if not errorlevel 61 cd \word\memos
        if errorlevel 59 if not errorlevel 60 cd \word\letters
        if errorlevel 0 if not errorlevel 59 goto RPLY_WRD
        word
        goto F3
37: :F2
38:     goto GET_RPLY
```

The line numbers in MAINMENU.BAT will change as you add commands (your text editor or word processor may not even show line numbers), but the labels remain accurate landmarks.

To add a sub-menu of your own, choose a function key and a program you use with several directories. Using the example as a guide, create the text file that contains the menu, change the function-key legend, and insert the commands in MAINMENU.BAT.

Testing your first sub-menu

Make sure that the sub-menu text file you created is in the current directory. To test your sub-menu, start the Main menu by entering:

```
C>mainmenu
```

You have tested the other selections, so press the function key that selects the sub-menu. The screen should clear and display the sub-menu. Press a function key on the sub-menu; the application program should start in the directory that corresponds to the choice you made from the sub-menu.

End the application program; the screen should display the sub-menu again. Press F10 to display the Main menu. Press F10 once again to return to DOS.

Tailoring the Menu System

You've seen how to add a program selection and a sub-menu to the Main menu. Now it's time to customize the menu for the programs that you use. But before you start, take the time to plan your selections and sub-menus. Sketch the Main menu with its 10 choices, and write in the selections you want. On the same piece of paper, write the names of the directories and

programs that each selection requires, so that you have all the information in one place when you start to revise the files.

Sketch the sub-menus you want in the same way. The most likely candidates for sub-menus are word-processing and spreadsheet programs, but you might include business graphics, database, telecommunications, CAD/CAM, or some other application program that uses several directories for data files. Again, type each selection and the names of the programs and directories that each selection requires.

It will be tempting to add all the selections at once. Resist this temptation. It takes a bit longer to add the menu selections one at a time, but it's much easier to test. It could take much longer if you filled out the entire menu all at once, then couldn't find one small, elusive error.

And don't forget to change the title lines of MAINMENU.DOC, PRNTMENU.DOC, and all the sub-menu DOC files, so that each menu announces that you didn't buy this menu system, you wrote it. The purpose isn't to amaze your friends—although a little bit of that might not hurt—but to remind you that you're in charge of this computer.

Adding Color to the Menus

The DOC files that contain the Main menu, Printer Setup menu, and Word-Processing menu include a few ANSI.SYS commands to highlight the titles. You can add more ANSI.SYS commands to make the screens more attractive, especially if you use a color display.

One particularly effective technique is to make each sub-menu selection on the Main menu a different color, then use that color for the selections on the sub-menu. However, it takes a lot of ANSI.SYS commands to make the function-key boxes different colors, so you might want to experiment with a copy of the DOC file until you decide how you want to use color in your menu system.

Remember, by the way, that if you add colors to PRNTMENU.DOC, you have to match those colors in PRNTOPT1.DOC and PRNTOPT2.DOC. Again, it might be prudent to experiment with copies of these files before you commit yourself to a particular use of color.

If you have defined your own standard colors as described in Chapter 11, the menus may not always look right. You can remedy that by inserting [E]*[m*—the ANSI.SYS command that turns off all display attributes—right after *@echo off* at the beginning of MAINMENU.DOC. To restore your standard colors (and your system prompt), make *promptrs* the last

command in MAINMENU.BAT, as described earlier in this chapter and in the section "Watch Out for Those Batch Files" in Chapter 11.

Both MAINMENU.DOC and WORDMENU.DOC in Appendix F include ANSI.SYS commands to set foreground colors.

Eliminating a Distraction

When a menu is on the screen, the cursor remains in the row and column position following the last item displayed. The cursor doesn't mark the location of something to be typed (there's nothing to be typed); it just blinks at you until you press a function key. This is another minor aesthetic blemish—like the system prompt flashing between the Main menu and the Printer Setup menu if you use the Command command—that you can correct, if you wish, with a few commands. If it doesn't bother you, skip the next paragraph.

To make the cursor invisible, insert the command *nocurs* after the label :START in MAINMENU.BAT (line 2 of Figure 15-2). To restore it, insert the command *normcurs* after the label :END in MAINMENU.BAT (line 45 of Figure 15-2). If you added *promptrs* as the last command to restore the system prompt, make sure it remains the last command in the file because PROMPTRS.BAT returns you to DOS.

Starting the Menu System with One Key

Even though the menu system handles most of your routine work with application programs, you'll need to return to DOS to copy or erase files, display directories, and take care of miscellaneous or infrequent chores. Rather than typing *mainmenu* each time you want to start the menu system again, you can assign the Mainmenu command to a key.

Alt-F10 is a fairly easy combination to enter; its key code is 0;113. To use a different key, substitute a different key code in the following command. Enter the following to assign the menu-system command to Alt-F10:

```
C>prompt $e[0;113;"mainmenu";13p
```

Now press Alt-F10. The screen should display the Main menu; press F10 to return to DOS.

To make this assignment permanent, put an Echo command that makes the key assignment in AUTOEXEC.BAT. To start the menu system with Alt-F10, as the previous example did, put the following command in AUTOEXEC.BAT.

```
echo [0;113;"mainmenu";13p
```

If you have the IBM enhanced keyboard (it has 12 function keys instead of 10), Alt-F12 is easier to type. Its key code is 140, so you would substitute 140 for 113 in the Echo command.

And if you're using DOS 5 and the Doskey program, you can assign a one-key macro to *mainmenu* with the following command:

```
C>doskey m=mainmenu
```

Then, if you type M and press Enter, the Mainmenu program runs. Refer to Chapter 13 for more information on Doskey.

Using the Menu System with a RAM Disk

If you don't have a RAM disk program, you can skip this section and go on to "A Complete Sample Menu System." If you have a RAM disk program and haven't yet read Chapter 10, do so before continuing with this section.

Every time DOS runs a batch file or a program, it must load the batch file or the program from disk into memory. The menu system uses several batch files and programs; you can speed up the menu system substantially by running the menu system from your RAM disk. If you always use a RAM disk, you can run your menu system from the RAM disk by adding a few commands to AUTOEXEC.BAT and MAINMENU.BAT.

These suggestions use an environment variable; if you aren't familiar with their use, read "Using Environment Variables in a Batch File" and "Creating Your Own Environment Variables" in Chapter 8. The suggestions also assume that your RAM disk is drive D and your normal system disk is drive C; if your RAM disk and system disk have different drive letters, substitute the correct letters in the examples that follow.

First add the commands shown in Figure 15-6 to AUTOEXEC.BAT.

```
1:  @echo off
2:  echo [7m*** Copying Menu System files to RAM disk ***[m
3:  md d:\menu
4:  copy \menu d:\menu > nul
5:  set path=d:\menu;<remainder of your command path>
6:  set dr=d:
7:  echo [0;140;"d:mainmenu";13p
    <Existing AUTOEXEC.BAT commands go here>
8:  mainmenu
```

Figure 15-6. *Sample AUTOEXEC.BAT file that runs the menu system from a RAM disk.*

The commands added to AUTOEXEC.BAT copy the files to the RAM disk and guarantee that the menu system will run from the RAM disk:

- Line 2 tells you that the files are being copied to the RAM disk.
- Line 3 creates the directory MENU in the root directory of the RAM disk. This matches the MENU directory on the fixed disk.
- Line 4 copies the menu-system files to the RAM disk.
- Line 5 puts the menu-system directory on the RAM disk at the beginning of your command path; you may want it in some other position, but be sure to include it and not to include C:\MENU. Add the directories in your current command path after *d:\menu*.
- Line 6 creates an environment variable named DR and sets its value to *d:*. DR represents the drive from which the menu system runs; it is used in MAINMENU.BAT to change the current drive. If your RAM drive uses a different letter, substitute its drive letter for *d*. Don't forget the colon: It makes the value of DR a valid command to change the current drive.
- Line 7 assigns the menu-system command to Alt-F12. If another program uses Alt-F12, or if you prefer another key or do not have an enhanced keyboard, substitute a different key code in the Echo command.
- Line 8 starts the Mainmenu program. It runs from the RAM disk because the first directory in the command path is D:\MENU, and C:\MENU isn't included in the command path.

Put the existing commands in AUTOEXEC.BAT between lines 7 and 8 of Figure 15-6.

Changes to MAINMENU.BAT

To use your menu system with a RAM disk, you'll need to change the current drive. To do this add the following commands to MAINMENU.BAT:

- Insert *%dr%* after the label :START (line 2 of Figure 15-2). DOS replaces *%dr%* with the value of the environment variable DR, and AUTOEXEC.BAT sets DR to *d:*, so this command changes the current drive to D.
- Insert *c:* (or the drive letter of your system disk, if it's not C) after each label that represents a program selection that uses a directory or program on your hard drive. For example, if F8 is a program selection, as shown in Figure 15-3, you would insert *c:* after the label :F8 (line 25 in Figure 15-3).

- Insert *c:* (or the drive letter of your hard drive, if it's not C) after each label that waits for a reply to a sub-menu that uses a directory or program on your hard drive. For example, if F3 selects a sub-menu, as shown in Figure 15-5, you would insert *c:* after the label :RPLY_WRD.
- Insert *%dr%* after the label :END (line 45 of Figure 15-2).

The next time you start or restart DOS, the menu system should run much more quickly; you'll appreciate the speed and your disk drives will appreciate the rest.

What If You Don't Always Use a RAM Disk?

You may not always want to use a RAM disk; the purpose of RAMDRIVE.BAT (described in Chapter 10) is to make it easy for you to use whatever sort of RAM disk you like—including none at all. This uncertainty complicates things a bit for your menu system, because you can't count on the RAM disk always being there; the AUTOEXEC.BAT file shown in Figure 15-6 won't work properly if you haven't defined a RAM disk.

But if you use RAMDRIVE.BAT to change your RAM disk configuration, AUTOEXEC.BAT can set up your menu system correctly whether or not there's a RAM disk, because the root directory contains a file named RAMDRIVE.LOG if a RAM disk is defined. Add the commands shown in Figure 15-7 to AUTOEXEC.BAT.

```
1:    @echo off
2:    if not exist ramdrive.log goto NO_RAMDRIVE
3:    echo [7m*** Copying Menu System files to RAM disk ***[m
4:    md d:\menu
5:    copy \menu d:\menu > nul
6:    set path=d:\menu;<remainder of your command path>
7:    set dr=d:
8:    goto END
9:  :NO_RAMDRIVE
10:   set path=c:\menu;<remainder of your command path>
11:   set dr=c:
12: :END
13:   echo [0;113;"mainmenu";13p
      <Commands always to be carried out in AUTOEXEC.BAT go here.>
14:   mainmenu
```

Figure 15-7. *Sample AUTOEXEC.BAT file that runs the menu system with or without a RAM disk.*

AUTOEXEC.BAT now contains two alternate setup routines: One runs the menu system from a RAM disk if there is one defined, and the other runs the menu system from an actual disk if there isn't a RAM disk. Line 2 of AUTOEXEC.BAT selects the appropriate routine based on whether or not the root directory contains RAMDRIVE.LOG:

- Line 2 goes to the label NO_RAMDRIVE, which sets up for the hard drive if RAMDISK.LOG doesn't exist in the root directory.
- Lines 3–8 are carried out if RAMDISK.LOG is in the root directory (which means that there is a RAM disk).
  - Line 3 tells you that the menu-system files are being copied to the RAM disk.
  - Line 4 creates the directory MENU in the root directory of the RAM disk. This matches the MENU directory on the actual disk.
  - Line 5 copies the menu-system files to the RAM disk.
  - Line 6 puts the menu-system directory on the RAM disk at the beginning of your command path. You may want it in some other position, but be sure to include it and not to include C:\MENU. Add the directories in your current command path after *d:\menu*.
  - Line 7 creates an environment variable named DR and sets its value to *d:*. DR represents the drive from which the menu system runs; it is used in MAINMENU.BAT to change the current drive. If your RAM drive uses a different letter, substitute its drive letter for *d*. Don't forget the colon: It makes the value of DR a valid command to change the current drive.
  - Line 8 goes to the label :END, which carries out the commands in AUTOEXEC.BAT that are always carried out, whether or not there is a RAM disk.
- The commands in lines 10 and 11 are carried out if RAMDRIVE.LOG is not in the root directory (which means that there is not a RAM disk):
  - Line 10 puts the menu-system directory on the real disk at the beginning of your command path. You may want it in some other position, but be sure to include it and not to include D:\MENU. Add the directories in your current command path after *c:\menu*.
  - Line 11 creates the environment variable named DR and sets its value to *c:* (see the explanation of line 7). If your system disk isn't in drive C, substitute the correct drive letter.
- The commands in lines 13 and 14—and any other commands between them—are always carried out:
  - Line 13 assigns the menu-system command to Alt-F12. If another program uses Alt-F12 or if you prefer another key, substitute a different key code in the Echo command.

– Line 14 starts MAINMENU.BAT.

Put the existing commands in AUTOEXEC.BAT between lines 13 and 14 of Figure 15-7.

The commands that change the current drive are the same as those listed in a previous section, under "Changes to MAINMENU.BAT." If you already made those changes, you're through. If you didn't make those changes, return to that section and make them now.

Now you can use the menu system whether or not a RAM disk is defined. The next time you start or restart DOS, AUTOEXEC.BAT will automatically run the menu system from the proper disk.

A Complete Sample Menu System

Appendix F shows the contents of all the files needed for a menu system with sub-menus (including the Printer Setup menu). It doesn't explain the files line-by-line—that would be mostly a repeat of this chapter—but it gives you a complete set of files to use as a reference. By comparing the menus and their selections with the commands in the batch files, you can see how to handle just about any situation.

DOS Shell and Other Ways to Cheat

Building batch files is challenging and fun, and it gives you a better understanding of DOS and your PC. Going to all that effort is worth it when the results are exactly what you want and you have the time to work on the system. When you don't, a multitude of programs exist that give you menu power and more, including the DOS Shell program included with DOS starting with version 4.0.

Programs like DOS Shell contain more than simple program selections; they're an actual replacement for DOS and many of DOS's commands. DOS Shell is easier to use than DOS for most beginners. Intermediate to advanced users can add their own programs and create a menu and sub-menu structure that appears at the bottom of DOS Shell's window, as seen in Figure 15-8.

More information about using DOS Shell can be found in *Running MS-DOS, Getting Started with MS-DOS 5 and the* New *MS-DOS Shell,* and the DOS manual.

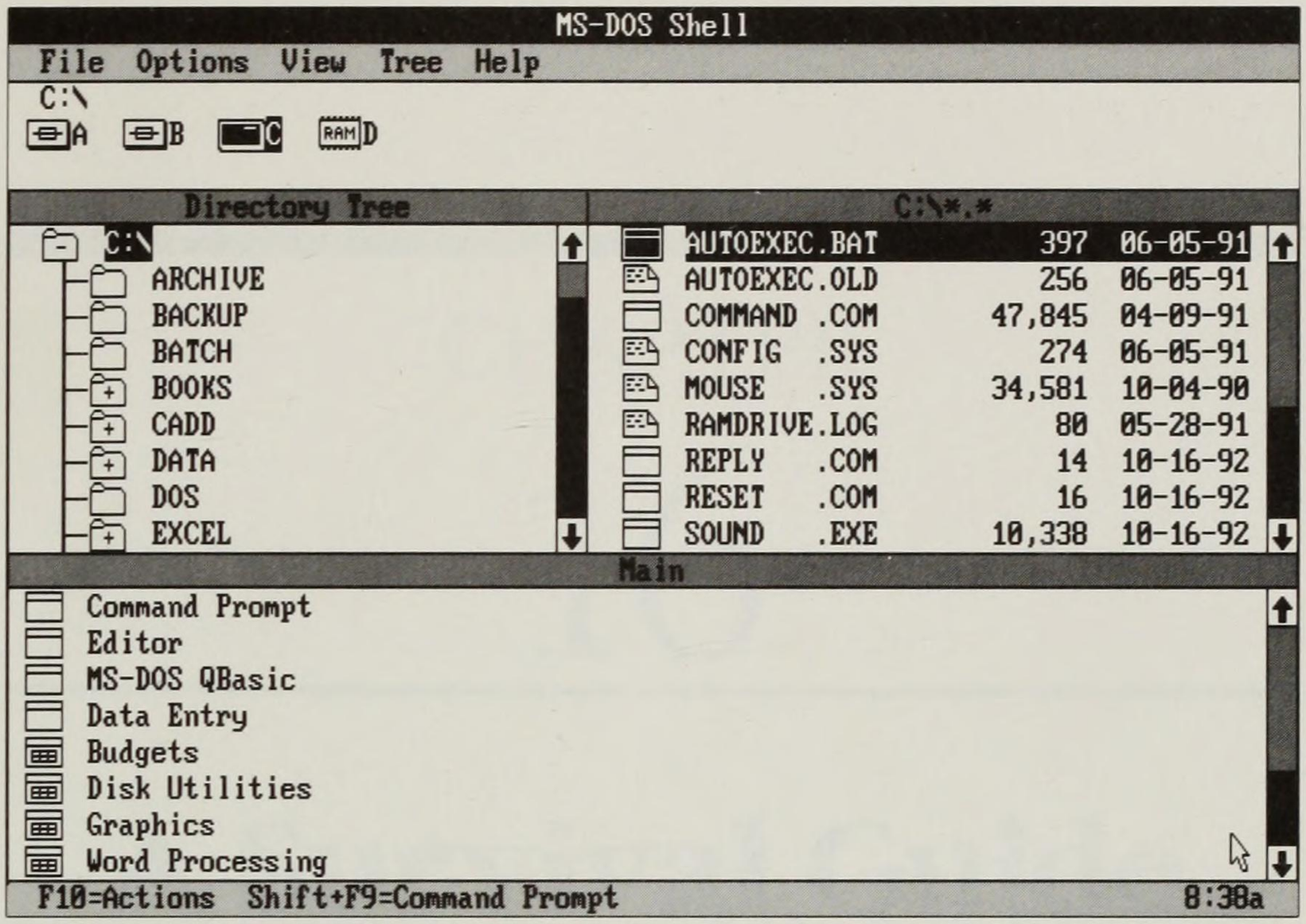


Figure 15-8. *The DOS Shell program allows you to create menus and sub-menus containing your programs.*

CHAPTER

16

A Survival Guide for the Nineties

DOS stands for Disk Operating System, which is really a misnomer. It's not an operating system at all. In fact, DOS is nothing more than a file manager—and a 10-year-old one at that. Today's computers run rings around the original PCs DOS was written for, yet we still use it.

As IBM and compatible personal computers mature in their second decade, solutions are continually developed for the problems created when you run an ancient, non-operating system on a modern 386 or 486 computer. Some would call these solutions BandAids—an ounce of prevention when a DOS cure is what's really needed. But they're not. These are survival tips—hints, tricks, and suggestions for squeezing the most power from DOS without sacrificing the power of your PC.

There is no single secret that will solve the problems of all DOS users. But the solutions covered in this chapter can be applied in a number of circumstances and benefit the widest possible range of users and situations. Most of these solutions involve DOS 5. If you haven't yet upgraded, save those parts of the chapter for when you do, or refer to your local software specialty store for individual solutions from third parties.

Preparing for Disaster

A computer running DOS is really a house of cards, ready to collapse at the slightest whisper of wind. Fortunately, most of the time we enjoy a stable computer environment. But when disaster does strike, you should be prepared for it. If you aren't, tools are available that will help undo some of the nasty things that computers (and their users) have been known to do.

The Emergency Boot Disk

An emergency boot disk is a good thing to have around, especially for hard-drive users. Floppy-disk systems already have boot disks; making an emergency copy is as easy as using the Diskcopy command. But hard-drive owners should think about creating a similar type of alternate startup disk.

An emergency boot disk must be a system disk. So start creating such a disk by putting a blank floppy into drive A and issuing the following command:

```
C>format /s
```

This Format command formats a system (bootable) disk in drive A. If the hard drive seizes, such a disk will help you get the system back up again.

An *emergency* boot disk should do more, so copy over the following files as well:

- Your CONFIG.SYS file. Additionally, you should copy any device drivers or external programs installed by CONFIG.SYS. This means setting up an appropriate subdirectory on the emergency boot disk, as well as editing the copy of CONFIG.SYS to reflect the file's new locations (on drive A).
- Your AUTOEXEC.BAT file. As with CONFIG.SYS, this requires copying any programs, utilities, startup programs, or memory-resident applications from the hard drive to an appropriate subdirectory on the emergency boot disk. However, you can be a little more judicious here; where CONFIG.SYS may require a device driver to run some necessary peripheral on your system, you may not require all the memory-resident tools and applications normally loaded by AUTOEXEC.BAT. But remember to edit AUTOEXEC.BAT to reflect any file's new locations on drive A.
- DOS's special file recovery and utility programs. These include Check Disk (Chkdsk), Debug, Backup and Restore, Fdisk, Format, Sys, and—with DOS 5—Undelete and Unformat.
- Important third-party utilities or tools. For example, if you're using the Norton Utilities, put a few of your favorite recovery tools, such as Unerase and the Disk Doctor, on your emergency boot disk.
- Backup logs. If you're using a third-party backup program, keep a second copy of your backup logs on the emergency boot disk. This will aid in the restoration and recovery of files in case anything truly nasty happens.

With the boot disk still in drive A, copy your AUTOEXEC.BAT and CONFIG.SYS files to it:

```
C>copy \config.sys a:
        1 file(s) copied

C>copy \autoexec.bat a:
        1 file(s) copied
```

Go through both files and look for device drivers, memory-resident software, and other utilities you use. For example, suppose the following is your CONFIG.SYS file.

```
device=c:\dos\himem.sys
dos=high
files=32
buffers=32
stacks=0,0
device=c:\dos\ansi.sys
device=c:\dos\ramdrive.sys 128 256 64
```

The device drivers here are HIMEM.SYS, RAMDRIVE.SYS, and ANSI.SYS. You probably don't need RAMDRIVE.SYS and ANSI.SYS for an emergency boot disk; copy them only if they fit. But HIMEM.SYS is important; copy it. In your own CONFIG.SYS file, decide which files and device drivers you'll need on your emergency boot disk.

With AUTOEXEC.BAT, suppose you have the following:

```
@echo off
path=c:\dos;c:\batch;c:\
prompt $p$g
set temp=c:\temp
doskey /insert
```

This is a limited example, but you can see that Doskey should be copied to the emergency boot disk. Any additional programs or utilities loaded by AUTOEXEC.BAT can be copied as well, but judge each utility by its importance; don't fill up the emergency boot disk with unnecessary files.

Now copy over DOS's special file recovery and utility programs. Most of these are located in your DOS subdirectory, so move there with *cd \dos* and then copy these files:

DOS 5.0	***Previous DOS Versions***
CHKDSK.EXE	CHKDSK.COM
DEBUG.EXE	DEBUG.COM
BACKUP.EXE	BACKUP.COM
RESTORE.EXE	RESTORE.COM
FDISK.EXE	FDISK.COM
FORMAT.COM	FORMAT.COM
SYS.COM	SYS.COM
UNDELETE.EXE	
UNFORMAT.COM	

In addition to the above, copy any third-party utilities, tools, or programs that will assist in analyzing and repairing a disk problem. Also, copy the most recent logs of any backups you've performed. Your emergency boot disk is almost complete.

It might not be possible to put all of these files on a single disk, especially if it's a 360K boot disk. If necessary, create two boot disks and keep as many files on them as possible. Once you're done, adequately label each disk and try it out: Put the emergency boot disk in drive A and press Ctrl-Alt-Del. This final step is required to make sure your system boots properly from the emergency boot disk. If not, check the CONFIG.SYS and AUTOEXEC.BAT files on that disk for any typing errors. Make sure the paths are all set to reflect files located on the boot disk and not the hard drive. Then reset again and make sure the boot disk works. It's better to find any errors now than when you really need the disk in the future.

The object here is to create a disk from which you can start your computer. Once that's done, you want as many tools, utilities, and helpers available to help you analyze and fix whatever problems you may have with the rest of the system. Set aside your emergency boot disk for now. Add to it later if you feel like it, especially to update software, device drivers, or backup logs. DOS 5 users will want to keep the disk handy for another exercise later in this chapter.

When Disaster Strikes

The first approach to dealing with a computer problem is to see just how far you get in the boot process. Did the computer come on at all? If not, it's probably a power supply problem. In addition to checking the main power plug (a classic, comical, yet common mistake) check all your connections: computer to wall, monitor and keyboard to system unit, and so on. Make sure the brightness isn't turned all the way down on the monitor.

Next, make sure the boot disk is working. This is primarily for a hard-drive system. Can you hear the hard disk spinning? Is the drive being accessed at all? Consider the messages your computer is sending: *Insert DOS disk in drive* x *and strike ENTER when ready*, *Non-system disk or disk error*, or *Invalid COMMAND.COM, insert COMMAND.COM disk in drive* x *and strike any key when ready*. They'll all stop your heart, but they usually indicate a bad boot disk. This could also be caused by a floppy disk you inadvertently left in drive A, so check that as well. Otherwise, it's a problem with the hard drive.

A frequent cause for a bad boot disk is missing files. Overenthusiastic beginners have been known to accidentally delete COMMAND.COM during feverish disk-cleaning campaigns. Even the "pros" who use the Shell configuration command to move COMMAND.COM will forget a subdirectory name, which gives the appearance of a bad boot disk. And with DOS version 5, using the Dir /a command (or the DOS Shell) you can see the

secret MS-DOS files and then erase them. It's at this stage in the game that you use your emergency boot disk. Follow these steps:

1. Turn off the computer and wait about 15 seconds.
2. Turn the system back on, this time inserting your emergency boot disk into drive A. Close the drive door.
3. Try to access drive C. Type *c:* and press Enter. If nothing happens, you could have a hardware error; it's best to have a professional look at it. If you can access the drive but files are missing, you may need to restore or rebuild the drive. Skip ahead to "The three R's," or if you have a hard-drive recovery utility, refer to its manual for directions and assistance.
4. If the boot drive is there, then check the boot files; look for COMMAND.COM. If it's gone, copy it from drive A by typing *copy command.com c:*. Then reset again to try to boot from the hard drive. If this doesn't work the invisible files are probably gone. You restore them using the System (*sys*) command as follows:

```
A>sys c:
```

   Enter *dir c:* to verify that COMMAND.COM is also there after you enter the System command; if it is not, copy it over with the following command:

```
copy command.com c:
```

5. If the boot disk boots, but files are missing, your job is to get the files back. Of course, this could be the rumblings of a major disk tremor that will hit sometime in the future. No disk is perfect, and performance will degrade over time. It's possible to recover lost files and do magic with disk utilities, but when the problem persists, consider buying a new hard drive.

The three R's

DOS comes with three utility programs that assist in recovering lost files from a damaged disk. They are Replace, Recover, and Restore.

DOS's R-word tools don't work miracles. The first, Replace, isn't even a data-recovery tool in the true sense; it's a utility that will scan one disk for files that exist on another. When a match is found, Replace deletes the older of the two files, replacing it with the newer. This is how DOS is updated, and Replace can also be used with other applications that lack sophisticated installation programs. For data recovery and working with a defective disk, however, this tool can't help much.

The second R-word tool is Recover, which has a miraculous sounding name but is a deleteriously nasty utility. Basically, Recover is designed to rebuild a file that happens to overlay a bad sector of the disk. It does this with ease but little grace: The file is renamed FILE*xxxx*.REC and the missing or bad piece is replaced with empty bytes or zeros. If a file were a snake, Recover would sew in a sausage for the missing piece and still try to convince you it was all snake. Use Recover only as a last resort, and then only on a floppy-disk copy of the file.

In truth, the best file-recovery utility you have (prior to DOS version 5.0) is Restore. Of course, this utility makes you feel good only if you've recently used its companion, Backup.

Consider the case of the single fouled file. If you've done a daily backup, you can get that file back with the following:

restore *drive filename*

drive is the disk drive containing the first backup floppy; *filename* is the full path name of the file you want to restore. After entering the command, Restore will successively scan through your backup floppy disks looking for *filename*. It's inelegant, but it works. (Note that if you use the BAKUP.BAT program discussed in Chapter 14, you can use the Copy command instead of Restore.

To recover a group of files, specify the proper wildcards in *filename*. And if you want to recover a portion of a subdirectory structure, tack on the /s switch.

Prior to DOS version 5.0, Restore is also your only way out of the ultimate disaster: the complete erasure of a hard drive.

If everything goes, you have to start over. You reformat the hard drive, then re-install DOS and all the files from your most recent backup. There isn't a DOS user alive who hasn't done this personally or been in the presence of someone who has. If the backup is recent, all you've lost is time. (This is why the Restore program is copied to the emergency boot disk and the Replace and Recover commands are not.)

Preventive Medicine

Besides screaming "Back up, back up, back up," the only real preventive medicine for your file system is the Mirror program included with DOS version 5. If you haven't yet upgraded, skip ahead to the section "Windows and DOS 5."

The Mirror program's job is to assist with data recovery. Mirror itself doesn't restore any data, but using Mirror will help other DOS 5 tools recover deleted files and trashed disks.

Mirror helps you prevent and deal with three different types of disasters. For each of those three, there is a complementary utility that remedies the situation.

Disaster	***Repair Utility***	***Mirror's Role***
Disk format	Unformat	Mirror saves the disk's status in a file named MIRROR.FIL. This file is used to help unformat the disk after it's been formatted.
Major disk catastrophe	Unformat	Mirror saves a hard drive's partition information in a file named PARTN-SAV.FIL. This file can be used to rebuild a hard disk that's been reformatted or damaged.
File deletion	Undelete	Delete tracking. In this mode, Mirror goes memory resident, keeping a constant eye on files you delete and saving information about them in a file named PCTRACKR.DEL.

If you use the proper options and switches, Mirror creates up to three files that are used by three other utilities to help remedy disk disasters. The following sections describe how to use the Mirror command in each situation; later sections in this chapter discuss the repair utilities mentioned in the above table.

Using Mirror to help unformat disks

When you reformat a disk using DOS's Format command, the disk itself is not erased. Usually, only the disk's boot information (or "sector"), the root directory, and the disk's File Allocation Tables (FAT) are erased. The rest of the disk is only verified, which is why you see the Track/Cylinder or percentage gauge increment even though the disk isn't being formatted. (A new floppy disk out of the box is always formatted all the way through, but this isn't important because there's no information on it to erase.)

The Format command can get away with erasing only the first part of a disk because without that information, the rest of the disk is only so many bytes. The important thing to keep in mind here is that an accidental reformat doesn't spell disaster. Most of your files are still intact. As long as you can reconstruct the part that Format erased, you can rescue the reformatted disk.

In Mirror's first incarnation, it reads the first part of a disk—the part Format erases—and saves all that information in a file named MIRROR.FIL. The Mirror command for this function is as follows:

mirror *drive*
or
mirror *drive* /1

drive is the name of the drive containing a disk you want to protect against accidental formatting. It must be followed by a colon. Mirror will save the boot sector, root directory, and FAT information for that disk in a file named MIRROR.FIL, which will be located in the root directory of that disk.

/1 (one, not a lowercase L) is a switch that tells Mirror to keep only one copy of MIRROR.FIL on the disk. If you don't specify /1, Mirror will create a backup file, MIRROR.BAK, when you run the Mirror command a second time on that disk.

To test Mirror, format a new *test* disk in drive A with the command *format a:*. Then create a file on that disk. This file (and the test disk) will be used in later examples in this chapter. Enter:

```
C>copy con a:test.doc
Erase me! Del me! Kill me!
And I keep coming back!
^Z
        1 file(s) copied
```

Enter *dir a:* to verify that the file is there. Now save the disk's information by entering the following command:

```
C>mirror a:

Creates an image of the system area.

Drive A being processed.

The MIRROR process was successful.
```

The Mirror command creates MIRROR.FIL. That's it! The important disk information is now saved in that file and can be used to unformat the disk at a later date. If you make any extensive modifications to the disk, you can run the Mirror command again. In fact, it's a good idea to run Mirror each time you start your computer by putting the proper command into your AUTOEXEC.BAT file, such as:

```
mirror c:
```

Above, Mirror is run on drive C, saving important information about that drive each time you start the computer. If you have more than one hard drive, you can specify more than one with a single Mirror command:

```
mirror c: d: e:
```

Above, Mirror creates one MIRROR.FIL file each for drives C, D, and E. The file for each drive is kept in that drive's root directory. Note that you shouldn't run Mirror on a drive created with the Subst command, nor is there any point in running Mirror on a RAM disk.

Remember, running Mirror is only the prevention; later in this chapter, you'll reformat the disk in drive A and rescue it completely using the Unformat command.

Obligatory warning: Never rely on this method as your only safety net. If you think, "Oh, heck, I can format this unlabeled disk because Mirror and Unformat will get the disk back for me should I discover that it's crucially important." This is wrong thinking, almost like saying, "I just had a tetanus shot, so now I should go out and step on some rusty nails." Always be careful. Only if disaster strikes should you rely on Mirror and Unformat. Even then, you should work with backups and safety copies of files before attempting to unformat.

Using Mirror to remedy major disk fits

Using Mirror to create a MIRROR.FIL file will help you resurrect a reformatted disk. But for major hard-drive catastrophes, MIRROR.FIL isn't going to be much help. In addition to the boot sector, a hard drive also has a *partition table*, which tells the operating system how much of the disk it can use and how to use it. If anything happens to the partition table, the entire disk is lost and must be reformatted at the lowest level. This is highly destructive and will ruin your whole day.

The Mirror command has an optional switch that will save a copy of your hard drive's partition information. You need to do this only once, unless you create a new partition or repartition your drive. The format is:

```
mirror /partn
```

/partn is the partition switch, telling Mirror to scan all hard drives in your system and save their partition information. The file created, PARTNSAV.FIL, is placed on a floppy disk.

If you created an emergency boot disk earlier in the chapter, retrieve it now and put it into your A drive. Enter *mirror /partn* at the DOS prompt, and follow the instructions on the screen:

```
Disk Partition Table saver.

The partition information from your hard drive(s) has been read.

Next, the file PARTNSAV.FIL will be written to a floppy disk. Please
insert a formatted diskette and type the name of the diskette drive.
What drive? A
```

The cursor is flashing under the *A*. Make sure your emergency boot disk is in drive A and press Enter, or type the letter of the correct drive and press Enter. Mirror responds with the message *Successful.* Verify that the file has been created. Enter *dir a:* and look for the file named PARTNSAV.FIL. If it's not there, repeat the *mirror /partn* command or refer to your DOS manual. Otherwise, put your emergency boot disk away for safekeeping. You won't need to update the PARTNSAV.FIL file again unless you use the Fdisk command to repartition your drive in the future.

The Mirror command, too, is only preventive. Later in this chapter, you'll read about the Unformat command, which uses the information in PARTNSAV.FIL to reconstruct a damaged hard drive. As usual, consider this a last resort. A recent backup is always your first, best, line of defense.

Using Mirror to help undelete files

Mirror's final effort at preventive medicine is its *delete-tracking* mode. Here Mirror becomes a memory-resident program that monitors all files you delete, saving information about them in a hidden disk file named PCTRACKR.DEL. Later, with the Undelete command, you can rescue deleted files more easily using the PCTRACKR.DEL file. Otherwise, undeleting files may not be fully successful.

The Mirror command with delete tracking uses the following format:

mirror /t*drive-files*

drive is the letter of the disk drive you want to monitor for delete tracking. It's not followed by a colon. If you want delete tracking for more than one drive, each drive letter must be specified after its own /t switch. For example:

```
mirror /tc /td /te
```

The above command turns on delete tracking for drives C, D, and E.

files indicates the number of files you wish delete tracking to monitor. *files* is optional and can range from 1 through 999. If you don't specify a value for *files*, Mirror uses 25 through 303, depending on the size of the specified disk drive. When specified, *files* must follow a hyphen after the drive letter.

To use Mirror with delete tracking, make sure your sample disk (created earlier in this chapter) is in drive A and enter the following command at the DOS prompt:

```
C>mirror /ta

Creates an image of the system area.

Drive A being processed.

The MIRROR process was successful.

Deletion-tracking software being installed.

The following drives are supported:
Drive A - Default files saved.

Installation complete.
```

The above command establishes delete tracking for drive A. If you'd also like to try delete tracking on drive C, add */tc* to the above command. The output of the command will reflect the change.

In addition to establishing delete tracking for a given drive, the Mirror command in this format will also write the disk information for the current drive to the MIRROR.FIL file. Above, drive C was processed. Note that if you don't specify the /1 switch, the backup file, MIRROR.BAK, will be created.

Once installed, Mirror with delete tracking will monitor all files deleted from the disk. When a file is deleted, information about the file will be saved in the PCTRACKR.DEL file. From there, you can undelete the file using the Undelete command, discussed later in this chapter. But as with the other implementations of the Mirror command, don't become over-confident. Using Mirror with delete tracking isn't an excuse for getting sloppy with the Del or Erase commands.

You can remove Mirror's delete-tracking ability from memory by using the /u, or unload, switch. The format is:

mirror /u

After entering the preceding command, Mirror will attempt to unload or remove delete tracking from memory. If it's successful, you'll see the message *Deletion-tracking software removed from memory.*

Formatting a Disk

The Format command has more switches and optional formats than any other DOS command. DOS 5 added two new switches to Format's already formidable array: the /q and /u switches. /q is used to quick-format an already-formatted floppy disk:

format *drive*: /q

drive is the letter of the drive containing a disk to be quick formatted. *drive* must be followed by a colon.

A quick format is just like a regular reformat of a disk. The only difference is that DOS will not verify the rest of the disk after it rewrites the boot sector, root directory, and FAT. This is a handy way to clean up a disk that may contain no-longer-needed files and subdirectories; it's faster than using the Format command without the /q switch. Note, however, that you cannot quick-format a disk to a different size; you must use the full Format command with the proper switches for that operation.

Retrieve the test disk you created earlier in this chapter, and put it into drive A. Verify that the disk's vital information has been saved by entering *dir a:* and looking for the file MIRROR.FIL. (If you've added to the disk since creating that file, use the Mirror command again; enter *mirror a:*.) If the MIRROR.FIL file is on that disk, you can proceed.

With your test disk in drive A, enter the following command:

```
C>format a: /q
Insert new diskette for drive A:
and press ENTER when ready...
```

The /q switch tells Format to quick-format the disk, which only erases the disk's boot information, root directory, and FAT. Because this is a test disk created for this purpose, it's okay; in the next section, you'll use the Unformat command to unformat the disk.

Press Enter. You'll see the following.

```
Checking existing disk format.
Saving UNFORMAT information.
QuickFormatting xxxx.
Format complete.

Volume label (11 characters, ENTER for none)?
```

The *xxxx* above will be replaced by the disk's size. Enter a volume label or simply press Enter. Enter *N* when asked *QuickFormat another (Y/N)?*.

The disk is now reformatted and empty. Prove it by entering *dir a:*. You'll see the classic error message *File not found*. The disk really is empty as far as DOS is concerned. But there are two layers of protection: The MIRROR.FIL file and DOS's built-in protection, the UNFORMAT information. Don't do anything else with this disk; you'll unformat it in the next section.

The /u switch is nearly the opposite of the /q switch: It's used to format and totally erase an already-formatted disk:

format *drive* /u

drive is the letter of the drive containing a disk to be formatted. *drive* must be followed by a colon.

The u in the /u switch stands for *unconditional*. DOS will reformat an already-formatted disk, overwriting all information on that disk. A new boot sector, root directory, and FAT will be created. After this operation, there is no way a disk can be recovered using the Unformat command—even if the Mirror command was used on that disk.

Obviously, you should use the /u switch with care. It's a drastic approach but one that may be necessary should you want a disk totally erased. For security reasons, a disk containing highly sensitive information should always be bulk-erased using a powerful magnet. But *format /u* should work well for all non-top-secret situations.

Unformatting a Disk

Only a few years ago, formatting was that final fatal act, an unforgivable sin you committed against your disk and data. With recent developments, however, the impossible is now possible: You can resurrect your data from the ashes of formatting destruction.

Starting with DOS version 5, the Unformat command lets you recover files and information from a reformatted or damaged disk. In some cases, a complete recovery is possible. The sooner you use the Unformat command on a reformatted or damaged disk, the better.

The Unformat command isn't a panacea: It's an emergency, last-ditch, last-resort, final attempt to save your disk. You should never rely completely on Unformat. Always check disks before you reformat them. Sure, knowing about Unformat will make you sleep better at night. But it's not an excuse to become negligent.

The format of the Unformat command has lots of options and switches:

unformat *drive* /j /l /test /partn /p /u

drive is the disk drive containing a disk to be unformatted and rebuilt. It must be followed by a colon.

/j is a test switch; specifying it will not unformat the disk. When you do use the /j switch, the Unformat command will cross-compare DOS's system information with that information saved by the Mirror command.

/l (lower case L, not one) when used with the /partn switch displays the disk's partition information, provided Mirror was run with its /partn switch on the same disk. (This is mainly used for rebuilding hard drives.) When used without the /partn switch, /l tells DOS to search the disk for files and assume that there is no MIRROR.FIL file present.

/test is a test switch; specifying it will not unformat the disk. /test works like /j but uses only DOS's system information. Use this switch when you haven't used Mirror on a disk.

/partn is used to restore a hard drive's partition-table information—but only when specified without the /l switch and when the disk's partition-table information has been saved using *mirror /partn.*

/p causes the Unformat command to echo all its messages to the printer.

/u forces Unformat to rebuild the disk using only DOS's information instead of the information in MIRROR.FIL.

These switches can make your head spin. A few of them are only for test situations, and you probably won't use them. Restore the floppy disk in drive A now by entering the following:

```
C>unformat a:

Insert disk to rebuild in drive A:
and press ENTER when ready.
```

The disk is in the drive, so press Enter. You'll see the following:

```
Restores the system area of your disk by using the image file
created by the MIRROR command.
```

```
    WARNING !!        WARNING !!

This should be used only to recover from the inadvertent use of
the FORMAT command or the RECOVER command. Any other use of the
UNFORMAT command may cause you to lose data!  Files modified since
the MIRROR image file was created may be lost.

Searching disk for MIRROR image.

The last time the MIRROR or FORMAT command was used was at 08:34
on 01-28-92.
The prior time the MIRROR or FORMAT command was used was at 08:32
on 01-28-92.

If you wish to use the last file as indicated
above, press L. If you wish to use the prior
file as indicated above, press P. Press ESC
to cancel UNFORMAT.
```

The times and dates you see on your screen will be different, but you still want to use the last (most recent) MIRROR.FIL file. Press L. You'll see:

```
The MIRROR image file has been validated.

Are you sure you want to update the system area of your drive A (Y/N)?
```

The Unformat command is telling you it found MIRROR.FIL on the floppy disk. Press Y. You'll see:

```
The system area of drive A has been rebuilt.

You may need to restart the system.
```

That last message, *You may need to restart the system*, applies only if you're rebuilding a boot disk or hard drive. For the floppy disk in drive A, it's not important. In fact, the disk has been recovered. Enter *dir a:* to verify it, or better still, enter *type a:test.doc*.

Mirror is persistent and cautious; it's not something to be taken lightly—just as formatting a disk isn't to be taken lightly. Remember these key points:

1. Use the command *mirror drive* to put recovery information for the Unformat command on the disk in the drive or drives represented by *drive*. Update the MIRROR.FIL file created by the Mirror command often, at least every time you start your computer or once a day.

2. Quick-format floppies when you reformat them.
3. For your hard drive, create the PARTNSAV.FIL file using *mirror /partn*, and keep that file on your emergency boot disk. If you need to rebuild a hard drive with that file, the command is *unformat /partn*. Pray you'll never have to use it.

Note that the Unformat command will attempt to rescue a reformatted disk without the MIRROR.FIL file. In such a case, your output will be different, it will take much longer, and DOS may not be 100 percent successful at rescuing the disk.

Undeleting Files

When you delete a file, DOS does two things: First it replaces the first letter of the file name in the directory with character E5 (hexadecimal). Second, it deletes the file's location and size from the FAT. Physically, the file still exists on disk. Some operating systems, in addition to removing the file name from the directory and the file allocation table, will actually go out to the disk and scrub the file's bytes away. But DOS doesn't, which is why it's possible to undelete a file.

Like the Unformat command, Undelete works best if you're using the Mirror command. In this case, it's the Mirror command's delete-tracking feature that makes file recovery a snap. Refer to the previous section, titled "Using Mirror to help undelete files," for the details.

The format of the Undelete command is:

undelete *filename* /dt /dos /list /all

filename is the name of the file to undelete. A group of files can also be rescued by using wildcards. If *filename* is omitted, Undelete will assume you want to restore all the erased files in the current directory.

/dt and /dos are two switches that tell Undelete how to find delete files. Use /dt to force Undelete to access Mirror's delete-tracking file, PCTRACKR.DEL. Use /dos only if you haven't used Mirror and there isn't a PCTRACKR.DEL file available.

/list shows you which files Undelete can recover, though no files will be recovered.

/all causes Undelete to rescue all possible files. Unlike Undelete without the *filename* option, /all will not prompt you before it recovers a file.

Put your test disk into drive A if it's not already there. Log to drive A with *a:* and enter the following:

```
A>undelete /list

Directory: A:\
File Specifications: *.*

    Deletion-tracking file not found.

    MS-DOS Directory contains    0 deleted files.
    Of those,    0 files may be recovered.

Using the MS-DOS Directory.

No entries found.
```

The above listing simply tells you that Undelete has nothing to do; all files are present and accounted for. If your display looks different, then you probably had other files on the disk that were deleted.

Now delete something. Make sure you're logged to drive A. Enter *del test.doc*. Silent and quick, DOS kills the file. Now try to recover it. Enter:

```
A>undelete test.doc

Directory: A:\
File Specifications: TEST.DOC

    Deletion-tracking file not found.

    MS-DOS Directory contains    1 deleted files.
    Of those,    1 files may be recovered.

Using the MS-DOS Directory.

    ?EST     DOC       53  1-28-92  8:32a  ...A  Undelete (Y/N)?
```

Type Y to rescue the file. DOS will ask you for the first letter of the file-name. Type T and you'll see *File successfully undeleted.* It lives!

The best approach to undeleting files is to use Mirror's delete-tracking ability. Activate it for drive A by entering the following:

```
A>mirror /ta-10
```

The switch /ta-10 causes Mirror to create a PCTRACKR.DEL file capable of storing only the 10 most recently deleted files for drive A. That's all you'll need for this test, although in real life you'll want a higher value.

Now delete the file TEST.DOC again. This time, Mirror's memory-resident portion will monitor the file as it goes, saving vital information in the PCTRACKR.DEL file:

```
A>del test.doc
```

Deleting the file takes longer because information about it is being saved. But notice how quickly the file is rescued after you enter the following command:

```
A>undelete test.doc

Directory: A:\
File Specifications: TEST.DOC

    Deletion-tracking file contains    1 deleted files.
    Of those,    1 files have all clusters available,
                 0 files have some clusters available,
                 0 files have no clusters available.

    MS-DOS Directory contains    1 deleted files.
    Of those,    1 files may be recovered.

Using the Delete Tracking file.

      TEST     DOC       53  1-28-92  8:32a  ...A  Deleted:  1-28-92  9:30a
All of the clusters for this file are available. Undelete (Y/N)?
```

You can see how much smarter this is; Undelete now knows the file's full name, as well as more information about the files deleted and what can and can't be restored. Press Y to bring the file back.

If you're going to undelete a file, do it quickly. If you add another file to disk, or even use the pipe (¦) with a DOS command, the deleted file's data may be overwritten. (The *clusters* will be used by other files.) If so, full recovery isn't possible. Then again, it's much better to be careful with the Del and Erase commands in the first place. Add extra insurance by running Mirror with its delete-tracking option.

DOS and the Future

Survival guides don't deal only with disaster. More often than not, learning to survive simply means being able to adapt to changing circumstances. For PC users, that means dealing with the computer's new and interesting capabilities.

Using Windows

Microsoft Windows is a graphical user environment for DOS computers. It's come a long way, evolving from a simple DOS shell into a powerful graphics extension to DOS. Windows helps exploit the full potential of your PC, as well as offering beginners and heavy graphics users an easy-to-use, highly productive, and integrated computing environment. It is most efficient when running Windows applications—programs written explicitly for the Windows environment—but it can also enhance non-Windows (DOS) applications.

Windows' modes of operation

Starting with version 3.0, Windows has three modes of operation:

- 386 enhanced mode
- Standard mode
- Real mode

Windows is really three programs in one and uses whatever system resources you have to achieve the highest possible mode.

386 enhanced mode

Windows runs best in 386 enhanced mode, which requires a 386 or later PC with at least 2MB total memory—640K conventional and 1MB extended. (386 enhanced mode can be run with only 1MB of memory, but this is not advised.) The more memory, the better; but it must be extended memory. You should remove any extended-to-expanded memory drivers, as Windows will make any necessary conversions internally.

In 386 enhanced mode, Windows is able to run Windows programs simultaneously. In addition, Windows can run standard DOS applications simultaneously with Windows and other DOS applications. This is referred to as *multitasking*, and it's made possible by a combination of the 386's built-in abilities and Windows' software.

If you're using DOS 5 and running Windows in enhanced mode, you should put the file WINA20.386 in your root directory. Because both Windows and DOS 5 use the 386 microprocessor's *A20* line, this file lets Windows know it's okay to run in enhanced mode. Otherwise, Windows would assume some other program is in charge and might malfunction.

Standard mode

Standard mode is the mode Windows runs in when you have a 286 PC with at least 256K of extended memory available or a 386 PC with less than

2MB of memory. In this mode, Windows programs will multitask and you can run DOS applications—but you cannot multitask them with Windows applications or other DOS programs.

Real mode

Real mode is Windows' simplest mode of operation. Windows runs in real mode on an 8088 PC, on a 286 or 386 PC with less than the required memory, or under an extended DOS environment, such as DESQview. (Refer to the section "DOS and the Extended DOS Environments" later in this chapter.) Also, some older Windows applications will run only in real mode. Real mode can also be run at any time by starting Windows with its optional /r switch.

In real mode, Windows is at its least impressive. It still functions, but Windows programs don't run simultaneously, and everything else seems sluggish. Serious Windows users will have a 286, or better, a 386 with lots of extended memory installed. A math coprocessor also helps. If you lack any of these, there are ways to boost Windows' performance using some of its special support files, many of which are similar to those offered with DOS version 5.

❑ **NOTE:** *To find out in which mode Windows is running, bring up the Program Manager window. Drop down the* Help *menu, and select the last item,* About Program Manager. *In the dialog box that appears, the current mode will be named in the middle, right above the amount of free memory.*

Increasing Windows' performance

To get the best performance from your hardware, Windows comes with an array of device drivers and support programs. Some of these are specific to 286 and 386 systems; others will benefit all PCs, and not just when you run Windows.

To speed up disk operations, Windows comes with its own RAM drive and disk-caching software. (RAM drives and disk caches are covered in Chapter 10.) SMARTDRV.SYS is used to set up a disk cache, which greatly improves Windows' performance, and RAMDRIVE.SYS sets up a RAM drive. These programs are nearly identical to those offered with DOS version 5.

SMARTDRV.SYS, the disk cache, is important because Windows interacts a lot with the disk—especially when you're low on memory. Adding more memory helps Windows directly. But setting up a disk cache with SMARTDRV.SYS is usually the best way to go.

When you install Windows, the Setup program may already create a disk cache for you. For example, you may see the following line in your CONFIG.SYS file:

```
device=c:\windows\smartdrv.sys 768 256
```

The above command creates a disk cache with a maximum size of 768 bytes and a minimum size of 256 bytes. Sizes will vary, depending on free memory. (For additional options for SMARTDRV.SYS, refer to "Speeding Up Disk Performance with a Cache" in Chapter 10 or to your Windows manual for more information.)

RAMDRIVE.SYS sets up a RAM disk for use with Windows. Copying some files to a RAM disk, or creating new files there, will speed up Windows:

```
device=c:\windows\ramdrive.sys 180 512 32 /e
```

The above command creates a 180K RAM disk with 512 bytes per sector and 32 entries available in the root directory. The /e switch tells DOS to create the RAM drive in extended memory. You can also create the RAM drive in expanded memory by using the /a switch in place of the /e switch. If you do not specify a memory switch, the RAM drive will be placed in conventional memory, which may reduce Windows performance. Refer to "Using RAMDRIVE.SYS in Other Releases of DOS" in Chapter 10 for more information.

Another way to increase Windows' performance is to set up a *swap file* if you are running in 386 enhanced mode. (See Figure 16-1.) A swap file is a file on your hard drive used to temporarily store information that normally resides in memory. This frees memory to speed up applications. The information is returned from the swap file to memory when needed. To do this, you force Windows to start in the real mode by typing *win /r*. In Windows,

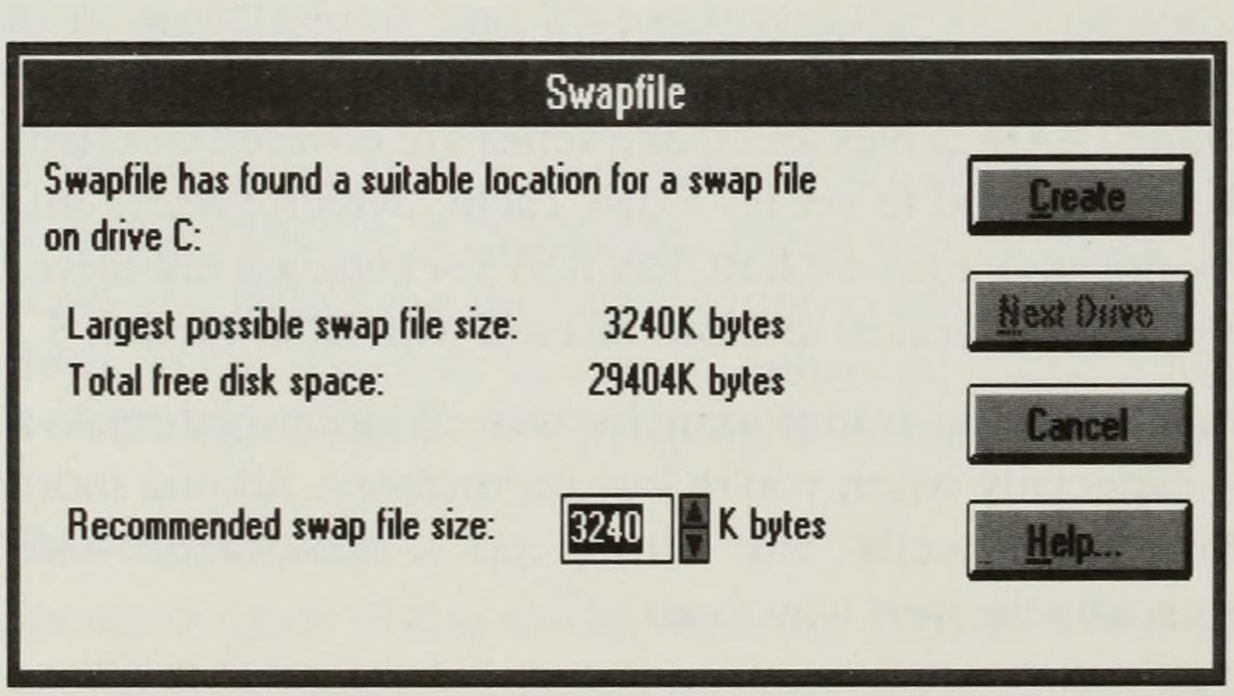


Figure 16-1. *Windows suggests a swap file size of 3240K bytes.*

close all applications (use the Task Manager to make sure only the Program Manager is running). Then select Run from the File menu and enter *swapfile*.

Create a swap file of the suggested size by clicking on the Create button. Or, if you feel you can spare the disk space, create a larger swap file. This will help speed disk operations, especially when memory is tight.

Windows can take advantage of the environment variable *temp*, which can be used to identify a subdirectory for temporary file storage. This is created, usually in a batch file, with the command *set temp=c:\temp*. A temporary subdirectory, such as C:\TEMP, is useful to keep around, and it's used by other programs and DOS version 5 as well. With Windows, you might want to assign the temporary variable to a RAM drive.

Windows and memory

Windows likes memory, but it's fussy depending on the mode it is running in. It doesn't just want more RAM, it wants more of a specific type of RAM, plus some Windows-specific device drivers to control that memory.

In real mode, applications can access only 640K of memory, but Windows can use the expanded memory for some of its own code storage. Windows also uses extended memory to swap information with conventional memory, but applications do not have access to any extended memory.

In standard mode, Windows applications can access as much extended memory as is available. Although Windows does not use expanded memory, it is made available to DOS applications that use it.

In 386 enhanced mode, Windows applications can access as much extended memory as is available but ignore any expanded memory. Expanded memory required by a DOS application is provided by an expanded-memory emulator.

To control extended memory, Windows uses an extended-memory system (XMS) device driver, HIMEM.SYS. HIMEM.SYS does many things for Windows. Above all, it establishes control over extended memory according to the XMS standard. Additionally, HIMEM.SYS gives Windows an extra 64K of DOS memory on PCs with at least 256K of extended memory. (HIMEM.SYS, XMS, and memory are covered in Chapter 9.)

If you have a 286 or later PC, Windows' Setup will automatically install the HIMEM.SYS device driver in your CONFIG.SYS file. If Setup locates any other XMS or extended device drivers in your CONFIG.SYS file, you'll be alerted to their presence and they'll be disabled. This is important: Windows knows what it works with best. There's no point in using anything less than the support software that came with Windows.

Windows can't run if you install a memory device driver that converts extended memory to expanded memory. However, if you do need extended memory to emulate expanded memory when you are not running Windows, use the EMM386.SYS device driver that comes with Windows. EMM386.SYS works only with 386 and later PCs. If you need expanded memory for 8088 or 286 PCs, you must have expanded memory installed.

Windows and EGA adapters

If you're using a PC with an EGA graphics display, you'll want to install the EGA.SYS device driver in CONFIG.SYS:

```
device=c:\windows\ega.sys
```

This will probably be done by the Setup program. EGA.SYS is required by Windows when you switch between the graphics mode and a text-only program in the real and standard modes. It helps Windows keep track of the EGA hardware, and makes switching between graphics and text modes run smoothly.

Windows and DOS 5

This section contains information about running Windows under DOS version 5. If you haven't yet upgraded to that version, skip ahead to the section titled "DOS and the Extended DOS Environments."

Windows and DOS 5 were made for each other. Actually, it would be more correct to say that a lot of time and effort has gone into making Windows and DOS version 5 work well together. In the previous section, you can see that many of Windows' memory-management feats, HIMEM.SYS, EMM386.SYS, the TEMP environment variable, and so on, have been incorporated into DOS 5.

If you're installing Windows on top of DOS 5, note the changes made to your CONFIG.SYS and AUTOEXEC.BAT files. Your original files will be saved with the extension OLD in place of SYS or BAT to allow you to make changes back, if needed. But what Setup does is relatively minor.

You'll notice that the Setup program uses Window's HIMEM.SYS and other device drivers, not DOS's drivers. This is the preferred plan of attack, according to Microsoft. In fact, it's always a good idea to use device drivers supplied with applications over those that came with DOS. This includes RAM disks, disk caches, and other device drivers as well.

❑ **NOTE:** *The exception to this rule is if you are installing Windows 3.0. The drivers supplied with DOS 5 are updated versions of the drivers and utilities that come with Windows 3.0; you should replace the Windows versions with the DOS versions.*

The only item worth concerning yourself with is the use of DOS 5's memory-management tricks with Windows. Moving DOS into the HMA, and device drivers and memory-resident software into upper memory blocks, gives Windows the largest possible memory space in which to operate. Fortunately, DOS 5 was created with Windows in mind, so this is easy.

If you need boning up on working with memory and using DOS 5's memory tools, refer back to Chapter 9, "Getting the Most from Memory." Also, note that these techniques work best on a PC with an 80386 or later microprocessor.

The first step to increasing Window's memory is loading HIMEM.SYS, which establishes the XMS standard and the HMA (High Memory Area). HIMEM.SYS is loaded automatically when you install Windows if you are using a 286 or later PC.

The second step is to move DOS into the HMA by putting the configuration command *dos=high* into your CONFIG.SYS file.

Device drivers and memory-resident software can be moved into the upper memory area and out of conventional memory, which frees even more space when you run Windows. Device drivers are relocated by specifying the *umb* option with the DOS configuration command, as in *dos=umb* (or, to combine this with the previous command, *dos=high,umb*). Device drivers can then be moved into the upper memory area by using the Devicehigh command instead of the Device command. For example:

```
devicehigh=c:\mouse\mouse.sys
```

To move memory-resident software, you need to load the EMM386.EXE file. The *noems* switch must be used, and zero EMS memory must be converted. The command *device=emm386.exe noems* in your CONFIG.SYS file will do the job. After that, use the Loadhigh command to relocate memory-resident software.

All of these techniques are covered in Chapter 9 in detail. If you normally run Windows in 386 enhanced mode but after trying these you note that the program is coming up in standard or real modes, that means not enough extended memory is available. Windows usually requires 2MB

total RAM to operate in 386 enhanced mode—and more extended memory is better. If you cannot get into enhanced mode, consider removing a RAM disk and reducing your disk cache to minimum size. Instructions are provided in the Windows manual.

DOS and the Extended DOS Environments

In addition to Windows, other DOS environments exist. These are collectively referred to as *extended DOS environments* because they augment and extend the basic power of regular DOS without changing DOS's appearance much.

DOS extenders work behind the scenes, giving DOS more power. An example of a DOS extender is the one built into Lotus 1-2-3 version 3, which allows 1-2-3 to directly access and use extended memory—a feat impossible for your typical DOS program. Other applications use similar built-in DOS extenders to access the advanced capabilities of your computer and still run under DOS.

An example of an extended DOS environment is DESQview. Built upon IBM's original multiple-application environment, Topview, DESQview gives DOS multitasking abilities, allowing you to run several DOS applications at the same time. Each application runs in its own "window," although it's not a graphics window as you have in Microsoft Windows.

The only drawback to extended DOS environments is that they won't run under Windows. While you can run 1-2-3 version 3, AutoCAD, and even Windows under DESQview, you cannot run extended DOS applications under Windows. The problem is confounded by conflicting device drivers: One set must be used when you boot Windows, and a different set must be used when you want to run an extended DOS environment.

Mastering Multiple CONFIG.SYS Files

Windows demands a specific system configuration that is set by commands in CONFIG.SYS. Those commands are often incompatible with extended DOS environments, which require CONFIG.SYS commands Windows doesn't like. It would be easy to solve this dilemma if all the files could exist in a single CONFIG.SYS file. But they can't. To accommodate more than one environment, you need several CONFIG.SYS files: one for Windows, one for DESQview, and so on.

DOS boots using a single CONFIG.SYS file, the one it finds in the root directory of your boot disk. Since only one file in the root directory can

have the name CONFIG.SYS, this limits your options. While you can't pick and choose commands in CONFIG.SYS, you can create multiple CONFIG.SYS files, one for each environment, and then switch among them before booting using one or more batch files. In the same way, you can create different AUTOEXEC.BAT files and switch among them as required by the environment you want to set up and use.

Consider two different situations:

1. You want to run Windows.
2. You want an extended DOS environment.

Each situation requires unique CONFIG.SYS and AUTOEXEC.BAT files. In practice, more situations could exist, but keep the number at two for the time being.

When you install Windows, the Setup program will create CONFIG.SYS and AUTOEXEC.BAT files customized for Windows, renaming your original files CONFIG.OLD and AUTOEXEC.OLD. Rename those files CONFIG.DOS and AUTOEXEC.DOS, because they were the original files you used simply to run DOS:

```
C>ren *.OLD *.DOS
```

If you elected not to save the original CONFIG.SYS and AUTOEXEC.BAT files, that's okay. It's easy to re-create them from the new files Windows created: First, copy CONFIG.SYS to CONFIG.DOS and AUTOEXEC.BAT to AUTOEXEC.DOS.

```
C>copy config.sys config.dos
        1 file(s) copied

C>copy autoexec.bat autoexec.dos
        1 file(s) copied
```

Using your text editor, word processor, Editor, or Edlin, edit the new DOS files:

1. Remove the Windows-only commands and device drivers from CONFIG.SYS. They should be easy to spot because they're all located in your WINDOWS directory. Note that Window's HIMEM.SYS file is placed in your root directory.
2. In AUTOEXEC.BAT, the only change Windows makes is to put your WINDOWS directory on the path. Edit the Path command and remove that directory if you like.

Save the files as CONFIG.DOS and AUTOEXEC.DOS.

You now have two sets of CONFIG and AUTOEXEC files:

Using Windows	*Using DOS*
CONFIG.SYS	CONFIG.DOS
AUTOEXEC.BAT	AUTOEXEC.DOS

Presently, the system will boot up using the Windows set of startup files. To change that and have DOS boot up using the original files, you need to use the Rename command four times:

```
C>ren config.sys config.win

C>ren autoexec.bat autoexec.win

C>ren config.dos config.sys

C>ren autoexec.dos autoexec.bat
```

Now the files have been switched. The set that boots up Windows is renamed with the WIN file-name extension; the old DOS set is renamed to the appropriate BAT and SYS extensions.

Using Windows	*Using DOS*
CONFIG.WIN	CONFIG.SYS
AUTOEXEC.WIN	AUTOEXEC.SYS

When you reset, DOS will now continue as it did before, ignoring the Windows startup files.

This situation isn't limited to DOS and Windows. If you're running other environments that have separate requirements, you can create unique system configuration files for them. You just need to rename them appropriately, then reset for the new configuration to take effect. Keep the extensions consistent and logical: WIN for Windows files and DOS for DOS files makes sense. Other extension names could be 386 for a system customized for the 386 environment, POD for Plain Old DOS, or DV for DESQview. Of course, using a batch file to control switching systems and resetting is the next logical step.

Entering the BOOT-BAT files

Figures 16-2 and 16-3 show the contents of two batch files, BOOT-WIN.BAT and BOOT-DOS.BAT.

Enter these files using your word processor, text editor, Editor, or Edlin. They're very similar, so you can create the first, copy it, then edit the copy to match the second file. Save each file under the appropriate name in your BATCH directory.

```
 1:    @echo off
 2:    echo Setting up to boot Windows...
 3:    c:
 4:    cd \
 5:    if exist config.dos goto OKAY
 6:    ren config.sys config.dos
 7:    ren autoexec.bat autoexec.dos
 8:    ren config.win config.sys
 9:    ren autoexec.win autoexec.bat
10:    echo CONFIG.SYS and AUTOEXEC.BAT have been renamed
11:    echo To reset,
12:    pause
13:    reset
14:  :OKAY
15:    echo Windows is already installed
16:  :END
```

Figure 16-2. *The BOOT-WIN.BAT file.*

```
 1:    @echo off
 2:    echo Setting up to boot Extended DOS...
 3:    c:
 4:    cd \
 5:    if exist config.win goto OKAY
 6:    ren config.sys config.win
 7:    ren autoexec.bat autoexec.win
 8:    ren config.dos config.sys
 9:    ren autoexec.dos autoexec.bat
10:    echo CONFIG.SYS and AUTOEXEC.BAT have been renamed
11:    echo To reset,
12:    pause
13:    reset
14:  :OKAY
15:    echo Extended DOS is already installed
16:  :END
```

Figure 16-3. *The BOOT-DOS.BAT file*

The following explanations describe how both batch files work. Because they're similar, the differences are noted where appropriate.

- Line 2 uses the Echo command to tell the user what the batch file is doing, configuring the computer to boot using either Windows' or DOS's CONFIG and AUTOEXEC files.
- Lines 3 and 4 log to drive C and change to the root directory.
- Line 5 verifies that the batch file needs to run. If the computer is already set to boot Windows (in Figure 16-2), then the file CONFIG.DOS will exist. Likewise, if the computer is set to boot Extended DOS (Figure 16-3), CONFIG.WIN exists. In either case, *goto OKAY* is executed since there's no need to rename any files or reset the computer.
- Lines 6–9 rename the CONFIG and AUTOEXEC files. The current files are renamed *.DOS or *.WIN, and the *.DOS or *.WIN files are renamed with the proper SYS or BAT extensions.
- Lines 10–11 display the message telling the user that the files have been renamed. The next step is to reset the computer, but before doing so, a Pause command is issued to give the user the chance to press Ctrl-C to cancel.
- Line 13 contains the RESET.COM program, which resets the computer. If you haven't yet created this program, see "Restart DOS with a Program" in Chapter 10.
- Line 14 is the OKAY label.
- Line 15 contains the message to be displayed if the computer is already set up with the proper CONFIG and AUTOEXEC files.

These two batch files only handle two different sets of CONFIG and AUTOEXEC files. It's conceivable that you could add a third, such as one that would run the computer without fancy memory drivers or DOS Shells—a "nude" CONFIG.SYS and empty AUTOEXEC.BAT. That complicates things, but it can still be done. You would add a second *if exist* test after line 5 to test for the third set of CONFIG and AUTOEXEC files with the proper extensions. This requires you to carefully think through which files need to be renamed what.

A risk you run when you play with multiple configuration files is losing your CONFIG.SYS and AUTOEXEC.BAT. It's true that DOS doesn't need either file to run. But if you've used the Shell configuration command to relocate COMMAND.COM (or a second command processor) away from the root directory, you'll see the *Bad or missing command interpreter* error message when you boot. If so, whip out your emergency boot disk and repair CONFIG.SYS or copy COMMAND.COM back to the root directory.

The boot-control system

Another approach to managing multiple CONFIG.SYS files is an overseer batch file, one that controls BOOT-WIN.BAT, BOOT-DOS.BAT, and any other configuration batch files you conjure up.

Figure 16-4 shows the BOOT.BAT program, which is used to oversee the other configuration batch files. Enter it using your word processor, text editor, Editor, or Edlin, and save it to disk as BOOT.BAT in your BATCH directory.

```
 1:    @echo off
 2:    cls
 3:    echo [7m Boot Control Center [m
 4:    echo Select a startup configuration
 5:    echo Escape to cancel
 6:    echo F1 - Boot Windows
 7:    echo F2 - Boot Extended DOS
 8:    rem (more Fkeys as needed)
 9: :REPLY
10:    reply
11:    rem (add Fkeys here)
12:    if errorlevel 60 if not errorlevel 61 boot-dos.bat
13:    if errorlevel 59 if not errorlevel 60 boot-win.bat
14:    if errorlevel 27 if not errorlevel 28 goto END
15:    goto REPLY
16: :END
```

Figure 16-4. *The BOOT.BAT file, which controls the other configuration batch files.*

The following describes how this batch file works:

- Line 2 clears the screen.
- Lines 3–7 set up the display, introducing the program with some inverse characters. (Refer to Chapter 3 for more information on the ANSI screen-control commands.) Lines 5–7 define the function keys used to select the different boot configurations and the Escape key, which is used to cancel.
- Line 8 is a place holder for more function keys (F3 and so on) that could be installed later if you create more boot configurations.
- Line 10 waits for a keypress. This line uses the REPLY.COM program, which was created in Chapter 7.

- Lines 12 and 13 interpret the key pressed: Line 12 tests for the F2 key, in which case the batch file BOOT-DOS.BAT is run; Line 13 tests for the F1 key, in which case the batch file BOOT-WIN.BAT is run. Notice how the keypresses match the function keys displayed in lines 6 and 7. Additional function keys can be added before line 12.
- Line 14 tests for the Escape key.
- Line 15 loops back to the REPLY label at line 9, limiting input to the function keys and the Escape key.
- Line 16 is the END label. The batch file only goes here if the Escape key is pressed.

Putting BOOT.BAT to work

To use BOOT.BAT you need your supplemental batch files that change the system configuration, BOOT-WIN.BAT, BOOT-DOS.BAT, and whatever other batch files you've created. Each of them, as well as RESET.COM and REPLY.COM, should be in a directory on the search path, such as \BATCH. BOOT.BAT can be put there as well.

To run BOOT.BAT, enter *boot* at the DOS prompt:

```
C>boot
Boot Control Center
Select a startup configuration
Escape to cancel
F1 - Boot Windows
F2 - Boot Extended DOS
```

Press F1 to boot Windows; the BOOT-WIN program sets up CONFIG.SYS and AUTOEXEC.BAT and then allows you to reset the computer. Press F2 to boot your Extended DOS environment; BOOT-DOS takes over and renames CONFIG.SYS and AUTOEXEC.BAT. Press Escape to leave your computer as is.

Note that BOOT.BAT doesn't use the Call command to run the secondary batch files. After all, if the batch file is successful, the computer resets; there will be nothing to return to!

This is just one way to gain control over a computer when you have multiple configurations. Professional *boot monitor* or *config juggler* programs let you set the configuration each time you reset your PC. For example, the program DBMON, for Dual Boot Monitor, is used in some versions of OS/2 to select OS/2 or DOS for starting your computer. It has the same effect as the three simple batch files covered in this chapter.

The Setver Command

Some DOS applications are *version sensitive*. When new DOS versions were coming out fast and furious in the late eighties, some software developers couldn't be sure that their applications would work under a new version of DOS, so they used the internal *DOS Version function* to make sure they were using the correct version of DOS. Of course, the DOS Version function is also used to make sure users of older DOS versions don't use newer software that might take advantage of DOS features not available in those older versions.

This whole mess was put into a box and tied with a ribbon, starting with DOS version 5. The Setver command is used to keep track of persnickety applications that require this or that version of DOS in order to run. Setver keeps an internal version table in low memory, listing unruly programs and the version of DOS under which each wants to run. When you run one of those applications under DOS 5, Setver fools it into thinking it's running under the appropriate version of DOS, and everyone is happy.

To use Setver, you first need to install the SETVER.EXE device driver into your CONFIG.SYS file. The format is:

```
device=c:\dos\setver.exe
```

There are no parameters other than the SETVER.EXE file in its subdirectory. The directory C:\DOS is assumed above.

Once the Setver device driver is loaded into memory, you use Setver as a DOS command to edit the internal version table. The format is:

setver *filename version*

or

setver *filename* /delete /quiet

filename is the path name of a program that runs under a specific version of DOS. It must be a COM or EXE file, and only one *filename* can be specified at a time (no wildcards).

version must follow *filename*, and it indicates the major and minor release numbers of the DOS version the program named in *filename* would like to run under. That information, both *filename* and *version*, is then stored in memory—as well as permanently on disk—so that DOS 5 will recognize the program and fool it into thinking it's running under a different DOS version.

/delete or /d is used following *filename* to remove that program from Setver's internal table. You should use this for any program upgraded to work under a later version of DOS.

/quiet is used following /delete if you want to suppress the output of the Setver *filename* /delete command.

When used without any parameters, Setver displays its current version table. Enter the following:

```
C>setver
```

A list of files will be displayed on your screen, each followed by a specific version number. If the list is too long, enter *setver* | *more* to see the complete version table.

To add a program to the version table, specify the program and its DOS version after *setver* on the command line. Enter the following:

```
C>setver fussy.exe 3.3
```

Setver doesn't check to see whether a file FUSSY.EXE exists. If you have another program running under DOS 5 that requires a previous version of DOS, specify its name and version number instead of *fussy.exe* and *3.3* above (use the same format). DOS displays the following message:

```
WARNING - The application you are adding to the MS-DOS version table
may not have been verified by Microsoft on this version of MS-DOS.
Please contact your software vendor for information on whether this
application will operate properly under this version of MS-DOS.
If you execute this application by instructing MS-DOS to report a
different MS-DOS version number, you may lose or corrupt data, or
cause system instabilities.  In that circumstance, Microsoft is not
responsible for any loss or damage.

Version table successfully updated
The version change will take effect the next time you restart
your system
```

This is a standard disclaimer, warning you that you're taking your life into your hands by running a version-specific application under DOS 5. Sometimes the application will run. But some programs, for example an old version of Hercules BASIC, want to run under one version of DOS only, and the Setver command can't fool them.

To put the changes into effect, reset by pressing Ctrl-Alt-Del.

Once your system has started again, use the Setver command to display the list of programs in the version table. Enter *setver*, and at the end of the list you will see:

```
FUSSY.EXE        3.30
```

When DOS runs FUSSY.EXE, it checks the internal version table, does a switch that tells Fussy it's working under DOS 3.3, and away you go. You hope.

When software is updated, you should remove it from the list. Enter the following:

```
C>setver fussy.exe /delete

Version table successfully updated
The version change will take effect the next time you restart
your system
```

Now FUSSY.EXE is removed from the list when you enter *setver*, but the change to the internal version table will take effect only after you restart the computer.

CHAPTER

17

The Care and Feeding of Your Computer

It wasn't so long ago that computers had to be isolated in their own expensive environments: temperature and humidity controlled within a narrow range, a false floor to keep all the cables out of harm's way. Occasional windows afforded fleeting glimpses of what went on inside the computer room, but the entrances were usually guarded. One of the more liberating qualities of personal computers is their tolerance of a more realistic environment. They require little more care than other electronic appliances, such as VCRs.

But there are still a few things to keep in mind. This chapter suggests ways to keep your computer healthy, and ideas on how to keep yourself comfortable while using it. It's not DOS, but it's all part of using a computer.

Pushing Back the Limits

You may have read that the uses for some piece of hardware or software are "limited only by your imagination." View any such claim with a severely jaundiced eye, not only because it raises the suspicion that the writer couldn't think of any good uses in time for deadline, but also because it just isn't true. The uses of your computer are limited, among other things, by the capacity and speed of its memory, disk drives, and microprocessor.

Capacity and speed

Too little memory or disk capacity may mean you can't even load a program, let alone run it; or maybe you can load it, but "run" is much too generous a term to describe its progress. More insidiously, a program may run just slowly enough to discourage you from making that last change to the spreadsheet or the report that's due this afternoon.

You're also limited by the reliability and availability of your system—it's hard to be productive on a system that occasionally lapses into psychotic behavior or on one that you have to share with other people. And sometimes you're limited simply by the number of hours in a day.

You may not be able to do much about sharing a system, and you can't change the number of hours in a day, but you can improve both the capacity and speed of your computer. If you haven't yet made the following improvements, each will help.

- Add memory, first to bring your system to the 640K maximum and then to give yourself extended or expanded memory. It shouldn't cost more than $200 for conventional memory, and may cost anywhere from $500 to $1,000 for extended/expanded memory, depending on how much you add. The more, the better.

- If you don't have a hard drive, buy one. You can get a 40MB drive for less than $350 if you shop aggressively. Figure your megabyte needs based on your applications' requirements. Take that value and multiply it by 3—that's the hard-drive size you need. You can add a second hard drive to most PCs, and it's often the best type of upgrade you can buy.
- Use a RAM disk, especially if you don't have a hard drive. If you use so many resident programs that you don't have enough memory for a RAM disk, consider eliminating enough resident programs to use at least a 64K RAM disk, or buy an expanded memory board.

Give the Big Red Switch a Rest

A computer, like a light bulb or a stereo amplifier, experiences a bit of extra electrical stress each time you turn it on. You can prolong the life of your computer by turning it on and off as infrequently as possible. Because computers require so little electricity—about the same as a 100-watt light bulb—leaving the system on doesn't cost much and will probably save you money by reducing maintenance costs.

Once you turn your system on, don't turn it off until you're through with it for the day. Don't, however, leave something displayed on the screen for hours at a time; this runs the risk of burning whatever is displayed into the phosphor coating on the inside of the screen. For the same reason that you don't turn the computer off, don't turn the display off; instead, turn down the brightness control or use a program that blanks the screen.

There's one exception to this practice: When you start using a brand-new computer, leave it on day and night for the first week or two. If an electrical component—especially an integrated circuit—is going to fail, chances are it will fail early in its life. If something is going to fail, you want it to fail as quickly as possible so it can be repaired while still under warranty.

Keep It Clean

Cleanliness is the best preventive maintenance you can perform on your computer. But don't be a fanatic about it; remember, personal computers are fairly hardy, so don't try for an antiseptic environment.

The normal home or office environment is usually fine for your computer, but do keep your floppy disks in their envelopes and store them in some sort of closed container. The stray dust mote is much more likely to garble the data on a floppy disk than to damage part of the computer.

Moving parts are most susceptible to dirt, and almost all the moving parts in your computer are located in one device: the keyboard. With rare exceptions, if your keyboard stops working, you stop working, so take some pains to keep it healthy.

The biggest enemy of keyboards is liquid, especially liquid that contains sugar. If you spill a soft drink, coffee, tea, or 100 percent–natural fruit juice on the keyboard, it will almost certainly stop working properly within a few days; maybe just a few keys will stick, fail to make contact, or otherwise misbehave, but the problem almost always renders the computer unusable. And unless you're exceptionally adept mechanically and have a good supply of small tools, there's a better-than-even chance that taking the keyboard apart to clean it will create even worse problems.

The only sure way to avoid this is never to drink while you're using the computer. But you probably will. So give yourself every chance: Find or make a place to put your glass, cup, or can at least a foot away from the keyboard. Never put it anywhere else. Put it down on your work, if necessary; better to spill something on that final copy you just spent an hour printing than on the keyboard. Don't put your drink above the keyboard: Put it below the keyboard if possible, on the same level if necessary, but never above. And look when you reach; if you keep your eyes glued to the screen (or the keyboard) while you reach for your drink, sooner or later you'll tip your drink over onto the keyboard.

If there's a noticeable amount of dust where you work, you can reduce the chance of a problem by putting a dust cover over the keyboard when you're not using it. One of those expensive, custom-fitted numbers isn't really necessary; a piece of plastic or tightly woven nylon will do nicely.

Smoking is far more dangerous to you than to your computer or disks. Contrary to the warnings you may have heard, computers, like old-time politicians, operate just fine in a smoke-filled room. But make it a habit to go to another room or, better yet, outside to smoke. You'll smoke less, the office or house will smell better, people won't yell at you for smoking, and your computer will stay cleaner. Everybody wins.

Static Electricity Is a Killer

But if the danger of dirt and smoke is exaggerated, the danger of static electricity may be understated. If your computer freezes or behaves erratically at random moments, the cause may be small static-electricity charges transmitted from you through the keyboard. If you sometimes feel a slight zap when you touch a doorknob or filing cabinet, you're carrying around enough static electricity to disturb or even destroy one or more of

the integrated circuits that make up your computer and to scramble data on a floppy disk.

Low humidity is the chief cause of static electricity (next to cats and thunderstorms); the condition is made worse by carpeting and clothing made of synthetic fibers. Low humidity is the norm in the mountain and desert states, but it can be a problem anywhere during the winter, especially in buildings with forced-air heating systems. It can be a year-round problem in a building whose climate-control system reduces the humidity too much.

Back to the symptom: If you feel a slight zap when you touch metal, the static-electricity situation is serious enough that you should do something about it. You can try two approaches: Dissipate any accumulated charge before you touch the computer, or reduce the potential for accumulating static electricity. You may want to try both.

Dissipating the charge is the more direct method. Develop the habit of touching something metal before you touch the computer or a disk, such as a chair frame, table leg, or lamp. If you're surrounded by plastic, put a substantial metal object—a large pewter ashtray, perhaps, or a bronze bowling trophy—near the computer and touch it. These methods are by no means foolproof, because the metal isn't grounded, but they provide substantially more protection than doing nothing. Static-dissipation devices are available that provide a grounded metal plate to touch; some attach to the keyboard, others are part of a power-distribution device that fits between the system unit and display. These units are more effective than merely touching something metal, and they may be necessary if static electricity is a serious problem in your workplace.

You can buy an antistatic mat—a small carpet with metallic threads woven through it. Be sure to get one large enough to cover the area where your chair moves around on the floor and the area under the table where you rest your feet. Antistatic sprays—to spray on your existing carpet—are also available, but the effect is temporary.

To reduce the accumulation of static electricity, you must increase the humidity, change or get rid of the carpeting, or both. Although a small room-sized humidifier may help, these steps often are too expensive to be practical. A less expensive (and correspondingly less effective) step is to avoid wearing leather-soled shoes or clothes made of synthetic fibers. (If possible, compute in the nude.) Whether or not you take any of these steps, you should still touch metal to dissipate the static charge before you touch the computer or a disk.

Power-line conditioners

The electric power from your wall plug isn't perfect; sometimes the voltage is a bit higher than it should be, sometimes it's a bit lower, and occasionally there might be a brief burst of high voltage called a spike. Most of these go unnoticed, but a severe irregularity can cause your computer to reset, perhaps losing some work. If this happens while the computer is reading from or writing to a disk, data on the disk can be damaged.

You can buy a device called a *surge suppressor* to smooth out the variations in the power, but seek some advice before buying one. Some of the cheaper ones don't offer much more than a false sense of security. And you may not need one at all; the power supplies in most IBM and IBM-compatible computers are remarkably tolerant of power fluctuations. If you have used your computer for several months and haven't noticed any errant behavior, chances are that you won't need one. But if there are large industrial users of electricity in your neighborhood, if a power-hungry device such as an air conditioner or refrigerator is on the same circuit as your computer, or if electrical storms are common in your part of the country, it might be a good idea to get a surge suppressor. A good one costs between $40 and $60.

A surge suppressor won't help when the power goes off unexpectedly. If you want protection from power outages, you need something that actually continues to provide electricity. Such a device is called an *uninterruptible power supply,* or UPS; it switches to a battery when the utility company power fails, and will run your computer for 10 to 20 minutes. A UPS isn't meant to be an alternate source of power; it simply keeps you running long enough to save files and shut the system down in an orderly fashion. UPS's are rated in watts; 200 watts should be sufficient for a floppy-disk system, and 300 or 350 watts for a system with a hard drive. A UPS of this capacity costs from $300 to $500.

If you want to keep working when the power is off, get both a UPS and a gasoline-powered 110-volt generator. The generator, too, is rated in watts. If you use it just for the computer, 400 watts is plenty. You can get one for less than $500. When the power fails, save your files and turn the system off. Unplug the computer from the UPS, start the portable generator, plug the computer into the portable generator, and turn the computer back on.

Do *not* plug the portable generator into the wall socket; you could send electricity back into the power lines and injure or kill a repair worker trying to fix the power lines.

Read the instructions carefully for a surge suppressor, UPS, or portable electric generator before you use it. If you have any questions about its

operation, ask the person who sold it to you. If you're still in doubt, especially about the portable generator, get the advice of an experienced electrician or your electric utility.

Treat Yourself Right Too

You spend as much time working in your office as the computer does, so don't neglect your own comfort and convenience.

Leave plenty of work space on either side of the keyboard. Keep a pad and pencil handy; no matter how many pop-up programs you have for taking notes and keeping track of telephone numbers and appointments, you're still going to want to write something down once in a while.

The proper distance and height of the display and keyboard vary from person to person, but take the time to try several positions until you find the most comfortable one for you. Make sure there are no windows or bright lights directly behind you or the monitor.

The quality of the lighting can have a dramatic effect on your comfort and efficiency. Bright overall lighting usually means glare on the display screen, and fluorescent lights often flicker; both situations can cause tension and eyestrain. Try to keep the lighting directed to your work area; swing-arm lamps are especially good, but watch out for heat from incandescent bulbs: It can damage floppy disks or even the plastic case of your computer. If you use swing-arm lamps, limit the bulbs to 60 watts. If you must have overall lighting, try to keep it indirect and not bright.

Put a telephone close by; you don't want to get up to answer the phone, and you'll probably consult with someone from time to time while you're sitting at the keyboard. If there isn't a phone jack close to the computer, use a long extension cord. If you use a modem, check whether it has two telephone jacks on the rear panel. If it does, plug the cord from the wall jack into one and the telephone into the other; if it doesn't, put a splitter on the end of the cord from the wall jack and plug one cord into the modem and the other cord into the phone.

Depending on the time of year, don't hesitate to open windows, turn on a fan or air conditioning, or turn on the heat to control the temperature. In most cases, if it's comfortable for you, your computer will be just fine.

Things to Buy (or Make)

If your system includes a printer and a color display, you have to turn three switches on or off (four if there's a lamp nearby). Save yourself the trouble, and save wear and tear on the switches, by plugging everything into a

multiple-outlet box. You can get a six-outlet box at a hardware store with a master switch and circuit breaker for $15 to $20; either attach it to the underside of your desk or work table, or just put it on the floor. If you wish, you can get a fancy console with an individual switch for each outlet that fits between the system unit and display for $100 or so. Some consoles include a surge suppressor; be sure the surge suppressor is a good one before spending the extra money.

If you're still putting a stack of fresh paper behind your printer and letting the finished pages fold up on top of the fresh, buy or make a printer stand. There isn't much to it—just something that holds the printer several inches above the tabletop to make room below for the stack of fresh paper. You can buy one for $15 to $30, or make one from a piece of ½-inch plywood and a couple of lengths of 1x4 or 1x6. (If you haven't thrown away your college furniture, you could use those bricks and boards.)

If you enter a lot of text or data, buy or make a copy stand. Again, it's fairly simple: just something to hold papers, magazines, or books at a comfortable reading angle. The adjustable kind on a spring-loaded arm are handy, but they cost $25 to $40 and won't handle anything very heavy. All you need is something that's about a foot high, with a front sloped at about a 15-degree angle, that you can keep from sliding under the weight of whatever you prop against it.

Buy one of those miniature vacuum cleaners that's about the size of a small flashlight. It's perfect for cleaning the keyboard, air vents, fans, disk drives, and other parts of the computer. Every three months or so—more often if your office has a lot of dust—disconnect the power cord and remove the cover from your computer and vacuum the inside. You should clean every surface you can see, but be careful not to damage any chips or connectors. This tool is great for cleaning cameras, calculators, VCRs, and other appliances and instruments. The vacuum cleaner goes through a 9-volt battery in less than an hour, so get the AC adapter too. You can buy both the vacuum cleaner and the AC adapter for less than $25.

Where Not to Save Money

For reasons described previously, don't turn your computer off each time you're going to be gone for an hour or two.

Don't buy bargain-basement floppy disks. The money you saved loses all meaning the first time you put an archive disk in drive A to do some more work on that masterpiece and DOS blandly responds *General Failure error reading drive A*. In the same vein, don't buy those disk punchers

that "magically" transform a 720K disk into a 1.4MB one; they don't work and can't work, so never count on them for reliably storing any data.

Don't save money on a chair. The more hours a day you spend at the computer, the more important it is to have a proper chair. Get one with good, high back support, and make sure it's deep enough to support the underside of your thighs all the way to the knee. The front edge of the seat should be rounded and padded well enough so that it won't cut off the circulation in your legs after several hours of sitting.

Don't waste much time trying to conserve printer paper. It's fairly cheap, and recycling centers love it; just separate it from the newsprint and cardboard, if you recycle those.

Keep at least one brand-new, name-brand printer ribbon stored away, because when you're ready to print the final copy of that masterpiece and discover that the ribbon in the printer is worn out, it's usually 3:30 a.m. or Sunday afternoon. Generic ribbons and re-inkers save substantial amounts of money, but that last re-inking may not have worked, and this generic ribbon may be the one that sneaked through Quality Control; spare no expense on your ace in the hole.

SECTION

III

Quick Reference

APPENDIX

A

A Quick Reference to ANSI.SYS Commands

Each ANSI.SYS command begins with [E][, the Escape character (code 27) followed by a left bracket.

The terms shown in *italics* in the following list are used in the command descriptions later in this appendix. They are grouped here to provide you with a quick reference.

attr specifies a display attribute such as high intensity, blinking, or color.

col is a number from 1 through 80 that specifies the column to which the cursor is to be moved.

cols is a number from 1 through 79 that specifies how many columns the cursor is to be moved.

keycode specifies the key to be defined. If the key is one of the standard ASCII characters, *keycode* is a number from 1 through 127. If the key is a function key, keypad key, or a combination of the Shift, Ctrl, or Alt key and some other key, *keycode* is two numbers separated by a semicolon, the first of which is always 0 and the second of which is taken from the table in Appendix D.

mode specifies the display mode. It is a number from 0 through 7.

result is the character or characters to be produced when a key is pressed. It can be specified as an ASCII code, an extended key code, a string enclosed in quotation marks, or any combination of codes and strings separated by semicolons.

row is a number from 1 through 25 that specifies the row to which the cursor is to be moved.

rows is a number from 1 through 24 that specifies how many rows the cursor is to be moved.

The following tables are in this appendix:

Figure	*Description*
A-1	Cursor commands
A-2	Erase commands
A-3	Display attributes and Mode commands
A-4	Keyboard commands

CURSOR COMMANDS

Cursor Up

ESC[*rows*A

Moves the cursor up the specified number of rows without changing the column.

rows is a number from 1 through 24 that specifies how many rows the cursor is to be moved up. If you omit *rows*, DOS moves the cursor up 1 row.

Examples

ESC[13A	Move the cursor up 13 rows.
ESC[A	Move the cursor up 1 row.

Cursor Down

ESC[*rows*B

Moves the cursor down the specified number of rows without changing the column.

rows is a number from 1 through 24 that specifies how many rows the cursor is to be moved down. If you omit *rows*, DOS moves the cursor down 1 row.

Examples

ESC[8B	Move the cursor down 8 rows.
ESC[B	Move the cursor down 1 row.

Cursor Right

ESC[*cols*C

Moves the cursor right the specified number of columns without changing the row.

cols is a number from 1 through 79 that specifies how many columns the cursor is to be moved right. If you omit *cols*, DOS moves the cursor right 1 column.

Examples

ESC[40C	Move the cursor right 40 columns.
ESC[C	Move the cursor right 1 column.

Cursor Left

ESC[*cols*D

Moves the cursor left the specified number of columns without changing the row.

cols is a number from 1 through 79 that specifies how many columns the cursor is to be moved left. If you omit *cols*, DOS moves the cursor left 1 column.

Examples

ESC[10D	Move the cursor left 10 columns.
ESC[D	Move the cursor left 1 column.

Figure A-1. *Cursor commands.* *(continued)*

Figure A-1. *continued*

Move Cursor	**ESC[*row*;*col*H or ESC[*row*;*col*f**	
	Moves the cursor to the specified row and column.	
	row is a number from 1 through 25 that specifies the row to which the cursor is to be moved. If you omit *row*, DOS moves the cursor to row 1. To omit *row* but specify *col*, enter the semicolon to show that *row* is omitted.	
	col is a number from 1 through 80 that specifies the column to which the cursor is to be moved. If you omit *col*, DOS moves the cursor to column 1.	
	If you omit both *row* and *col*, DOS moves the cursor to the home position (row 1, column 1—the upper left corner of the screen).	
Examples	ESC[;10H	Move the cursor to row 1, column 10.
	ESC[H	Move the cursor to row 1, column 1.
Save Cursor Position	**ESC[s**	
	Stores the current row and column position of the cursor.	
	You can later move the cursor to this location with a Restore Cursor Position command.	
Example	ESC[s	Save the current cursor position.
Restore Cursor Position	**ESC[u**	
	Moves the cursor to the row and column position most recently saved with a Save Cursor Position command.	
Example	ESC[u	Move the cursor to the row and column last saved with a Save Cursor Position command.

ERASE COMMANDS

Erase Display	**ESC[2J**	
	Erases the entire display (equivalent to the DOS Clear Screen [cls] command).	
Example	ESC[2J	Erase the screen.
Erase to End of Line	**ESC[K**	
	Erases from the current cursor position through the end of the line that contains the cursor.	
Example	ESC[K	Erase from the cursor to the end of the line.

Figure A-2. *Erase commands.*

Set Attribute

ESC[*attr*m

Turns on a characteristic, or *attribute,* of the display, such as high intensity, blink, or foreground and background color.

attr specifies the display attribute to be turned on. More than one attribute can be specified by using a semicolon to separate the attribute numbers. *attr* can be any of the following:

Attribute	*Color*	*Foreground*	*Background*
0 None	Black	30	40
1 High intensity	Red	31	41
4 Underline	Green	32	42
5 Blink	Yellow	33	43
7 Reverse	Blue	34	44
8 Invisible	Magenta	35	45
	Cyan	36	46
	White	37	47

If you omit *attr*, all attributes are turned off (equivalent to specifying *attr* as 0).

Examples

`ESC[1m`	High intensity.
`ESC[1;5m`	High intensity and blink.
`ESC[30;46m`	Black foreground, cyan background.
`ESC[m`	Turn off all attributes.
`ESC[0m`	Turn off all attributes.
`ESC[0;1;36m`	Turn off all attributes, then turn on high-intensity cyan foreground.

Set Display Mode

ESC[=*mode*h

Sets the width and color capability of the display (generally equivalent to the DOS Mode command). This command can also be used to cause lines longer than 80 characters to be broken at the eightieth character and continued on the next line, rather than truncated at the eightieth column; this is called *line wrap*. It can be turned off with the Turn Off Line Wrap command. Note the equal sign (=) that precedes *mode*.

mode specifies the display mode. It can be one of the following:

Mode	*VGA Mode*	*EGA Mode*	*CGA Mode*
0	360x400, 16 colors	320x350, 16 colors	320x200, monochrome
1	360x400, 16 colors	320x350, 16 colors	320x200, 4 colors
2	720x400, 16 colors	640x350, 16 colors	640x200, monochrome
3	720x400, 16 colors	640x350, 16 colors	640x200, 4 colors

Figure A-3. *Display attributes and Mode commands.* *(continued)*

Figure A-3. *continued*

	Mode	VGA Mode	EGA Mode	CGA Mode
Set Display Mode *(continued)*	4	320x200, 4 colors	320x200, 4 colors	320x200, monochrome
	5	320x200, 4 colors	320x200, 4 colors	320x200, 4 colors
	6	640x200, monochrome	640x200, monochrome	640x200, monochrome
	7	Turn on line wrap	Turn on line wrap	Turn on line wrap
	13	320x200, 16 colors	320x200, 16 colors	–
	14	640x200, 16 colors	640x200, 16 colors	–
	15	640x350, monochrome	640x350, monochrome	–
	16	640x350, 16 colors	640x350, 16 colors	–
	17	640x480, 2 colors	–	–
	18	640x480, 16 colors	–	–
	19	320x200, 256 colors	–	–

Examples `ESC[=1h` Set the display to 40 by 25, color on.

`ESC[=7h` Continue lines longer than 80 characters; don't truncate them.

Turn Off Line Wrap

`ESC[=7l`

Causes lines longer than 80 characters to be truncated at the eightieth character, rather than continued to the next line.

Example `ESC[=7l` Truncate lines longer than 80 characters.

NOTES: Modes 0, 1, 4, 5, 13, and 19 display 40 columns of text; all other modes display 80 columns. A dash (–) means the mode is not available on the adapter.

KEYBOARD COMMANDS

Define Key

`ESC[`*keycode*`;`*result*`p`

Assigns one or more characters to be produced when you press a key.

keycode specifies the key to be defined. If the key is one of the standard ASCII characters, *keycode* is a number from 1 through 127. If the key is a function key, keypad key, or a combination of the Shift, Ctrl, or Alt key and some other key, *keycode* is two numbers separated by a semicolon, the first of which is always 0 and the second of which is taken from the table in Appendix D.

result is the character or characters to be produced when a key is pressed. It can be specified as an ASCII code, an extended key code, a string enclosed in quotation marks, or any combination of codes and strings separated by semicolons. To restore a key to its original meaning, enter a Define Key command that sets *result* equal to *keycode*.

Figure A-4. *Keyboard commands.* *(continued)*

Figure A-4. *continued*

Examples	⎋[126;92p	Redefine the tilde (~) key as a backslash (\).
	⎋[126;126p	Restore the tilde (~) key to its original definition.
	⎋[0;112;"dir sort";13p	Redefine Alt-F9 as a Directory command piped to a Sort command, followed by a Carriage Return.
	⎋[0;112;0;112p	Restore Alt-F9 to its original definition.
Disable Enhanced Key Assignments	**⎋[0q**	Causes ANSI.SYS to ignore commands that reassign values to the additional keys on the enhanced keyboard, even if the DEVICE=ANSI.SYS command in CONFIG.SYS includes the /X parameter.
Enable Enhanced Key Assignments	**⎋[1q**	Causes ANSI.SYS to carry out commands that reassign values to the additional keys on the enhanced keyboard if the DEVICE=ANSI.SYS command in CONFIG.SYS includes the /X parameter.

NOTE: Keys that appear only on the enhanced keyboard—such as F11, F12, and the dedicated cursor-control keys—can be redefined only if the DEVICE=ANSI.SYS command in CONFIG.SYS includes the /X parameter and does not include the /K parameter.

APPENDIX

B

Epson-Compatible and HP-Compatible Printer Commands

This appendix briefly describes some of the more common commands accepted by Epson and Epson-compatible dot-matrix printers and by Hewlett-Packard and HP-compatible laser printers. Most printers accept more commands than are shown here; check your printer's manual for a complete list of the commands it accepts and the required dip-switch settings.

Figures B-1 through B-3 and B-4 through B-6 summarize three groups of printer commands for the Epson and HP printers:

- Typeface commands, shown in Figures B-1 and B-4, control the kind of type to be used, which may include normal, bold, extended, compressed, elite, and italic typefaces.
- Format commands, shown in Figures B-2 and B-5, control such characteristics as the amount of space between lines, the skip to a new line or page, page size, margins, and tabs.
- Printer control commands, shown in Figures B-3 and B-6, reset the printer to its standard settings, sound the bell, cause the paper-out detector to be ignored, set the direction and speed of printing, and control other machine-related functions.

The command descriptions use these conventions:

^*E* represents holding down the Ctrl key and pressing the key shown by the uppercase letter. Figure C-4 in Appendix C shows the codes and corresponding DOS echo for characters 1 through 31.

<*Alt-5*> represents holding down the Alt key and typing the digits on the numeric keypad (not on the upper row of keys on the main keyboard). <Alt-5> and ^E both represent character 5. No matter how you enter it, DOS echoes character 5 as ^E; your word processor or other programs might echo it as ♣.

<*character 0*> represents character 0 (zero). You cannot enter character 0 with the Alt-key technique; most printers accept Alt-128 in its place.

[E] represents character 27 (called *Escape*). You cannot enter this character with the Esc key; either use the Alt-key technique or, if you're using Edlin, enter Ctrl-V followed by a left bracket ([).

EPSON TYPEFACE COMMANDS

Boldface Start	⎋E	
Boldface Stop	⎋F	
Condensed Start	^O or <*Alt-15*>	
Condensed Stop	^R or <*Alt-18*>	
Double Print Start	⎋G	
Double Print Stop	⎋H	
Elite	⎋M	12 characters per inch
Expanded Start	^N or <*Alt-14*>	One line only
	⎋W1	Until stopped
Expanded Stop	^T or <*Alt-20*>	Stops ^N only
	⎋W0	Stops ⎋W1 only
	⎋W0	Stops both ⎋W1 and ^N
Italic Start	⎋4	
Italic Stop	⎋5	
NLQ (Near Letter Quality)	⎋n	
Pica	⎋P	10 characters per inch
Subscript Start	⎋S1	
Superscript Start	⎋S0	
Subscript/superscript Stop	⎋T	
Underline Start	⎋-1	
Underline Stop	⎋-0	

Figure B-1. *Epson typeface commands.*

EPSON FORMAT COMMANDS

Form Feed	^L or <*Alt-12*> Advances the paper to the top of the next page.
Left Margin	⎋l*col* Sets the left margin to the specified column. (Lowercase L, not number 1.)
Line Spacing 6 lpi	⎋2 Sets the line spacing to six lines per inch.
Line Spacing 8 lpi	⎋0 Sets the line spacing to eight lines per inch.
Line Spacing in 1/72"	⎋A*units* Sets the line spacing to the specified multiple of 1/72".
Line Spacing in 1/216"	⎋3*units* Sets the line spacing to the specified multiple of 1/216".
Page Length	⎋C0*inches* Sets the page length to the specified number of inches.

Figure B-2. *Epson format commands.* *(continued)*

Figure B-2. *continued*

Right Margin	⎋Q*col* Sets the right margin to the specified column.
Skip Perforation	⎋N*lines* Skips the specified number of lines at the bottom of each page. If you start printing at the top of a page, this leaves top and bottom margins equal to half the perforation skip.
Tab Set	⎋D*col*...<*character 0*>
	Sets a tab at the column specified by *col*, up to 32. Specify each tab with the character whose code is the column number: Column 1 is <*Alt-1*>, for example, and column 10 is <*Alt-10*>. End the list with <*character 0*>.
Tab Set (repeating)	⎋e<*character 0*>*cols*
	Sets a tab position every specified number of *cols* across the entire line. ⎋e <*character 0*> <*character 5*>, for example, sets tabs at columns 5, 10, 15, and so on. Most printers accept <*character 128*> as well as <*character 0*>. Some printers call this command *Set Tab Unit.*

EPSON PRINTER CONTROL COMMANDS

Bell	^G or <*Alt-7*>
	Sounds the beep on the printer
Reset	⎋@
	Resets the printer to its standard settings
Disable Paper-Out Detector	⎋8
	Makes the printer ignore the paper-out detector; necessary to print close to the bottom of single sheets of paper
Enable Paper-Out Detector	⎋9
	Makes the printer stop when it nears the bottom of a sheet of paper; prevents printing on the platen when you run out of continuous-form paper

Figure B-3. *Epson printer control commands.*

HP PCL TYPEFACE COMMANDS

Symbol Set	
PC-8	E(10U)
Roman-8	E(8U
Spacing (proportional)	E(s1P
Spacing (fixed)	E(s0P
Pitch (characters per inch)	E(s*pitch*H
Height	E(s*points*V
Style	
Upright	E(s0S (This turns off the styles mentioned below.)
Italic	E(s1S
Condensed	E(s4S
Compressed	E(s8S
Expanded	E(s24S
Outline	E(s32S
Inline	E(s64S
Shadowed	E(s128S
Stroke	
Weight	
Light	E(s-3B
Medium	E(s0B
Bold	E(s3B
Typeface Selection	
Line Printer	E(s0T
Pica	E(s1T
Elite	E(s2T
Courier	E(s3T
Helvetica	E(s4T
Times Roman	E(s5T
Underline Start	E&d0D
Underline Stop	E&d@

Figure B-4. *HP PCL typeface commands.*

HP PCL FORMAT COMMANDS

Form Feed	^L or <*Alt-12*> Ejects page from the printer.
Left Margin	E&a*col*L Sets the left margin to the specified column.
Line Spacing	E&l*val*D Sets the line spacing to *val* lines per inch.
Vertical Motion Index	E&l*units*C Sets the space between rows to the specified multiple of 1/48".

Figure B-5. *HP PCL format commands.* *(continued)*

Figure B-5. *continued*

Page Size	E&l*val*A Sets the size of the page: 2=letter, 3=legal.
Page Length	E&l*lines*P Sets the page length to the specified number of lines.
Right Margin	E&a*col*M Sets the right margin to the specified column.
Top Margin	E&l*lines*E Sets the top margin to the specified line.
Text Length	El*lines*F Sets the text length to the specified number of lines.
Horizontal Cursor Positioning	E&a*col*C Sets the cursor to column *col*, as measured from the first column, which is 1.
Perforation Skip On	E&l1L
Perforation Skip Off	E&l0L
Page Orientation	E&l*val*O A *val* of 0 indicates portrait orientation; 1 indicates landscape orientation.

HP PCL PRINTER CONTROL COMMANDS

Bell	^G or <*Alt-7*> Sounds the beep on the printer
Reset	EE Resets the printer to its standard settings; ejects a page

Figure B-6. *HP PCL printer control commands.*

APPENDIX

C

The ASCII and IBM Extended Character Sets

Figures C-1 and C-2 show the ASCII standard character set and the IBM extended character set: the 256 characters supported by most computers that run MS-DOS. These figures show the characters in four columns; each character is followed by its corresponding code in decimal and hexadecimal. Many compatible printers print the full character set; if you're not sure about your printer, check its manual. Figure C-1 shows the first 128 characters (codes 0 through 127) of the ASCII standard character set;

ASCII	Ctrl	Dec	Hex	ASCII	Dec	Hex	ASCII	Dec	Hex	ASCII	Dec	Hex
	^@	0	00	<space>	32	20	@	64	40	`	96	60
☺	^A	1	01	!	33	21	A	65	41	a	97	61
☻	^B	2	02	"	34	22	B	66	42	b	98	62
♥	^C	3	03	#	35	23	C	67	43	c	99	63
♦	^D	4	04	$	36	24	D	68	44	d	100	64
♣	^E	5	05	%	37	25	E	69	45	e	101	65
♠	^F	6	06	&	38	26	F	70	46	f	102	66
•	^G	7	07	'	39	27	G	71	47	g	103	67
◘	^H	8	08	(	40	28	H	72	48	h	104	68
○	^I	9	09	)	41	29	I	73	49	i	105	69
◙	^J	10	0A	*	42	2A	J	74	4A	j	106	6A
♂	^K	11	0B	+	43	2B	K	75	4B	k	107	6B
♀	^L	12	0C	,	44	2C	L	76	4C	l	108	6C
♪	^M	13	0D	-	45	2D	M	77	4D	m	109	6D
♫	^N	14	0E	.	46	2E	N	78	4E	n	110	6E
☼	^O	15	0F	/	47	2F	O	79	4F	o	111	6F
►	^P	16	10	0	48	30	P	80	50	p	112	70
◄	^Q	17	11	1	49	31	Q	81	51	q	113	71
↕	^R	18	12	2	50	32	R	82	52	r	114	72
‼	^S	19	13	3	51	33	S	83	53	s	115	73
¶	^T	20	14	4	52	34	T	84	54	t	116	74
§	^U	21	15	5	53	35	U	85	55	u	117	75
▬	^V	22	16	6	54	36	V	86	56	v	118	76
↨	^W	23	17	7	55	37	W	87	57	w	119	77
↑	^X	24	18	8	56	38	X	88	58	x	120	78
↓	^Y	25	19	9	57	39	Y	89	59	y	121	79
→	^Z	26	1A	:	58	3A	Z	90	5A	z	122	7A
←	^[	27	1B	;	59	3B	[	91	5B	{	123	7B
∟	^\	28	1C	<	60	3C	\	92	5C	¦	124	7C
↔	^]	29	1D	=	61	3D	]	93	5D	}	125	7D
▲	^^	30	1E	>	62	3E	^	94	5E	~	126	7E
▼	^_	31	1F	?	63	3F	_	95	5F	⌂	127	7F

Figure C-1. *The ASCII standard character set.*

Figure C-2 shows the IBM extended character set (codes 128 through 255). Figure C-3 shows the four sets of box-drawing characters in the IBM extended character set. Figure C-4 shows the names and codes for the *control characters*, the first 32 ASCII characters (codes 0 through 31).

ASCII	Dec	Hex	ASCII	Dec	Hex	ASCII	Dec	Hex	ASCII	Dec	Hex
Ç	128	80	á	160	A0	└	192	C0	α	224	E0
ü	129	81	í	161	A1	┴	193	C1	β	225	E1
é	130	82	ó	162	A2	┬	194	C2	Γ	226	E2
â	131	83	ú	163	A3	├	195	C3	π	227	E3
ä	132	84	ñ	164	A4	─	196	C4	Σ	228	E4
à	133	85	Ñ	165	A5	┼	197	C5	σ	229	E5
å	134	86	ª	166	A6	╞	198	C6	µ	230	E6
ç	135	87	º	167	A7	╟	199	C7	τ	231	E7
ê	136	88	¿	168	A8	╚	200	C8	Φ	232	E8
ë	137	89	⌐	169	A9	╔	201	C9	Θ	233	E9
è	138	8A	¬	170	AA	╩	202	CA	Ω	234	EA
ï	139	8B	½	171	AB	╦	203	CB	δ	235	EB
î	140	8C	¼	172	AC	╠	204	CC	∞	236	EC
ì	141	8D	¡	173	AD	═	205	CD	φ	237	ED
Ä	142	8E	«	174	AE	╬	206	CE	ε	238	EE
Å	143	8F	»	175	AF	╧	207	CF	∩	239	EF
É	144	90	▒	176	B0	╨	208	D0	≡	240	F0
æ	145	91	▓	177	B1	╤	209	D1	±	241	F1
Æ	146	92	▓	178	B2	╥	210	D2	≥	242	F2
ô	147	93	│	179	B3	╙	211	D3	≤	243	F3
ö	148	94	┤	180	B4	╘	212	D4	⌠	244	F4
ò	149	95	╡	181	B5	╒	213	D5	⌡	245	F5
û	150	96	╢	182	B6	╓	214	D6	÷	246	F6
ù	151	97	╖	183	B7	╫	215	D7	≈	247	F7
ÿ	152	98	╕	184	B8	╪	216	D8	°	248	F8
Ö	153	99	╣	185	B9	┘	217	D9	•	249	F9
Ü	154	9A	║	186	BA	┌	218	DA	·	250	FA
¢	155	9B	╗	187	BB	▓	219	DB	√	251	FB
£	156	9C	╝	188	BC	▄	220	DC	η	252	FC
¥	157	9D	╜	189	BD	▌	221	DD	2	253	FD
Pt	158	9E	╛	190	BE	▐	222	DE	▪	254	FE
ƒ	159	9F	┐	191	BF	▀	223	DF		255	FF

Figure C-2. *The IBM extended character set.*

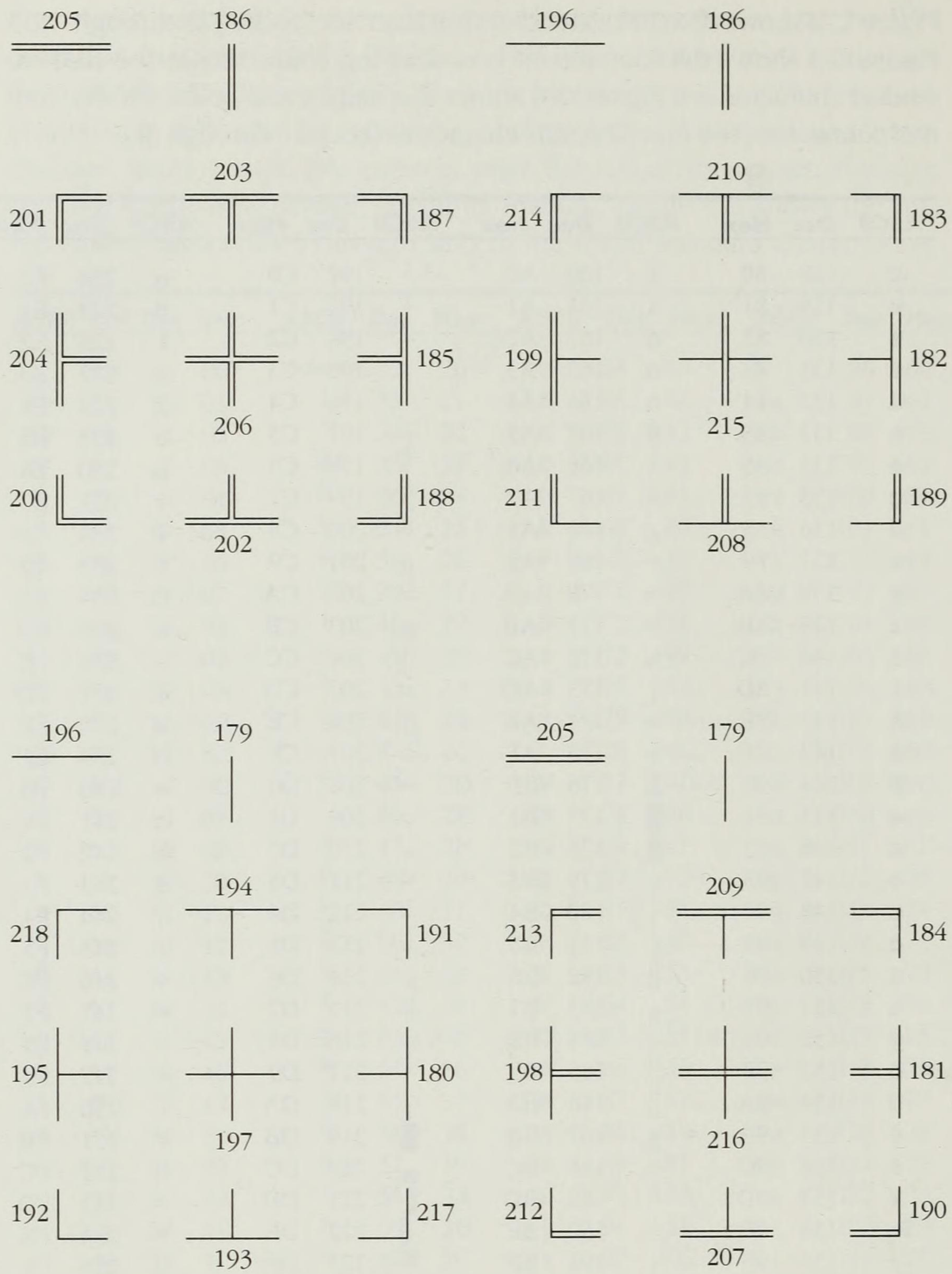


Figure C-3. *Box-drawing characters in the IBM extended character set.*

Abbreviation	Full Name	Code Dec	Code Hex	To Enter With Alt-Key	To Enter With Ctrl-Key	Echoes As
NUL	Null	0	0	none	none	nothing
SOH	Start of heading	1	1	Alt-1	Ctrl-A	^A
STX	Start of text	2	2	Alt-2	Ctrl-B	^B
ETX	End of text	3	3	Alt-3	Ctrl-C	^C
EOT	End of transmission	4	4	Alt-4	Ctrl-D	^D
ENQ	Enquiry	5	5	Alt-5	Ctrl-E	^E
ACK	Acknowledge	6	6	Alt-6	Ctrl-F	^F
BEL	Bell	7	7	Alt-7	Ctrl-G	^G
BS	Backspace	8	8	Alt-8	Ctrl-H	^H
HT	Horizontal tab	9	9	Alt-9	Ctrl-I	^I
LF	Linefeed	10	A	Alt-10	Ctrl-J	^J
VT	Vertical tab	11	B	Alt-11	Ctrl-K	^K
FF	Formfeed	12	C	Alt-12	Ctrl-L	^L
CR	Carriage return	13	D	Alt-13	Ctrl-M	^M
SO	Shift out	14	E	Alt-14	Ctrl-N	^N
SI	Shift in	15	F	Alt-15	Ctrl-O	^O
DLE	Data link escape	16	10	Alt-16	Ctrl-P	^P
DC1	Device control 1	17	11	Alt-17	Ctrl-Q	^Q
DC2	Device control 2	18	12	Alt-18	Ctrl-R	^R
DC3	Device control 3	19	13	Alt-19	Ctrl-S	^S
DC4	Device control 4	20	14	Alt-20	Ctrl-T	^T
NAK	Negative acknowledge	21	15	Alt-21	Ctrl-U	^U
SYN	Synchronous idle	22	16	Alt-22	Ctrl-V	^V
ETB	End transmission block	23	17	Alt-23	Ctrl-W	^W
CAN	Cancel	24	18	Alt-24	Ctrl-X	^X
EM	End of medium	25	19	Alt-25	Ctrl-Y	^Y
SUB	Substitute	26	1A	Alt-26	Ctrl-Z	^Z
ESC	Escape	27	1B	Alt-27	Ctrl-[	^[
FS	File separator	28	1C	Alt-28	Ctrl-\	^\
GS	Group separator	29	1D	Alt-29	Ctrl-]	^]
RS	Record separator	30	1E	Alt-30	Ctrl-^	^^
US	Unit separator	31	1F	Alt-31	Ctrl-_	^_

Figure C-4. *ASCII control-code names (characters 0 through 31).*

APPENDIX

D

ANSI.SYS Key and Extended Key Codes

Figures D-1 and D-2 show the identifying codes used with the ANSI.SYS Define Key command for all keys and key combinations that can be identified. The first column shows the key legend. The column headed *Alone* shows the code for pressing just the key; the next three columns show the code for holding down the key at the top of the column (Shift, Ctrl, or Alt) and pressing the key shown in the first column.

Where one number is shown, it is the ASCII code for the key; where two numbers are shown, they are the extended code whose first number is always 0. A dash (—) means the key or key combination can't be redefined. N/A marks combinations with the Shift key that can't be entered (Shift-colon, for example).

Entries in *italic type* identify the additional key combinations added to ANSI.SYS in version 4.0 of DOS. To use these extended key codes, you must use DOS 4.0. Make sure that you include the /x parameter with the Device command in CONFIG.SYS that names ANSI.SYS (for example, *device=\dos\ansi.sys /x*). If you're not using version 4.0, or you don't include the /x parameter, you can't identify these key combinations.

An extended code is two numbers, the first of which is always 0. Figure D-1 shows the extended codes for the function keys (F1 through F12), keypad keys, and PrtSc. Figure D-2 shows the single-number codes for the standard ASCII characters (letters, numbers, punctuation marks, other symbols, plus Backspace, Tab, Enter, and Esc) and the extended codes for key combinations.

The extended code for including a null character (ASCII 0) in the result code of the Define Key command is 0;3.

Key	*Alone*	*Shift*	*Ctrl*	*Alt*
F1	0;59	0;84	0;94	0;104
F2	0;60	0;85	0;95	0;105
F3	0;61	0;86	0;96	0;106
F4	0;62	0;87	0;97	0;107
F5	0;63	0;88	0;98	0;108
F6	0;64	0;89	0;99	0;109
F7	0;65	0;90	0;100	0;110
F8	0;66	0;91	0;101	0;111
F9	0;67	0;92	0;102	0;112
F10	0;68	0;93	0;103	0;113
F11	*0;133*	*0;135*	*0;137*	*0;139*
F12	*0;134*	*0;136*	*0;138*	*0;140*
Home	0;71	55	0;119	*0;151*
Cursor Up	0;72	56	*0;141*	*0;152*
Pg Up	0;73	57	0;132	*0;153*
Cursor Left	0;75	52	0;115	*0;155*
Cursor Right	0;77	54	0;116	*0;157*
End	0;79	49	0;117	*0;159*
Cursor Down	0;80	50	*0;145*	*0;160*
Pg Dn	0;81	51	0;118	*0;161*
Ins	0;82	48	*0;146*	*0;162**
Del	0;83	46	*0;147*	*0;163**
PrtSc	—	—	0;114	—
Keypad Enter	13	13	10	*0;166*
Keypad *	42	42	*0;150*	*0;55*
Keypad +	43	43	*0;144*	*0;78*
Keypad –	45	45	*0;142*	*0;74*
Keypad /	47	47	*0;149*	*0;164*
Keypad 5	76	53	*0;143*	*0;76*

*Alt-Ins and Alt-Del produce these key codes only when the Ins and Del keys in the cursor-control portion of the Enhanced keyboard are pressed, not the Ins and Del keys in the keypad portion.

Figure D-1. *Extended codes for function and cursor-control keys.*

Key	*Alone*	*Shift*	*Ctrl*	*Alt*
Backspace	8	8	127	0;14
Tab	9	0;15	*0;148*	*0;165*
Enter	13	13	10	0;28
Esc	27	27	27	*0;1*
'	39	N/A	_	0;40
,	44	N/A	_	*0;51*
-	45	95	_	0;130
.	46	N/A	_	*0;52*
/	47	N/A	_	*0;53*
0	48	41	_	0;129
1	49	33	_	0;120
2	50	64	_	0;121
3	51	35	_	0;122
4	52	36	_	0;123
5	53	37	_	0;124
6	54	94	0;30	0;125
7	55	38	_	0;126
8	56	42	_	0;127
9	57	40	_	0;128
;	59	N/A	_	*0;39*
=	61	43	_	0;131
[	91	N/A	27	*0;26*
\	92	N/A	28	*0;43*
]	93	N/A	29	*0;27*
`	96	N/A	_	*0;41*
A	97	65	1	0;30
B	98	66	2	0;48
C	99	67	3	0;46
D	100	68	4	0;32
E	101	69	5	0;18
F	102	70	6	0;33
G	103	71	7	0;34
H	104	72	8	0;35
I	105	73	9	0;23
J	106	74	10	0;36
K	107	75	11	0;37
L	108	76	12	0;38
M	109	77	13	0;50
N	110	78	14	0;49
O	111	79	15	0;24
P	112	80	16	0;25
Q	113	81	17	0;16
R	114	82	18	0;19
S	115	83	19	0;31

Figure D-2. *Extended codes for standard ASCII characters.* *(continued)*

Figure D-2. *continued*

Key	*Alone*	*Shift*	*Ctrl*	*Alt*
T	116	84	20	0;20
U	117	85	21	0;22
V	118	86	22	0;47
W	119	87	23	0;17
X	120	88	24	0;45
Y	121	89	25	0;21
Z	122	90	26	0;44

Converting Hexadecimal Numbers to Decimal

APPENDIX

E

Converting Hexadecimal Numbers to Decimal

Hexadecimal numbers, like decimal numbers (or numbers of any other base) represent powers of the base number—in this case, 16. The rightmost digit represents the instances of 16 to the zero power, or 1; the second digit from the right represents the instances of 16 to the first power, or 16; the third digit from the right represents the instances of 16 to the second power, or 256; the fourth digit from the right represents the instances of 16 to the third power, or 4096; and so on.

Figure E-1 simplifies converting hexadecimal numbers up to FFFF to decimal. Each section of the table lists the decimal value of each digit in the corresponding hexadecimal number.

Fourth Digit		***Third Digit***		***Second Digit***		***First Digit***	
Hex	*Dec*	*Hex*	*Dec*	*Hex*	*Dec*	*Hex*	*Dec*
0	0	0	0	0	0	0	0
1	4096	1	256	1	16	1	1
2	8192	2	512	2	32	2	2
3	12,288	3	768	3	48	3	3
4	16,384	4	1024	4	64	4	4
5	20,480	5	1280	5	80	5	5
6	24,576	6	1536	6	96	6	6
7	28,672	7	1792	7	112	7	7
8	32,768	8	2048	8	128	8	8
9	36,864	9	2304	9	144	9	9
A	40,960	A	2560	A	160	A	10
B	45,056	B	2816	B	176	B	11
C	49,152	C	3072	C	192	C	12
D	53,248	D	3328	D	208	D	13
E	57,344	E	3584	E	224	E	14
F	61,440	F	3840	F	240	F	15

Figure E-1. *Hexadecimal-to-decimal conversion table.*

To use Figure E-1 to convert a hexadecimal number to decimal, find the decimal value of each hexadecimal digit and add the decimal values to find the decimal equivalent of the hexadecimal number. For example, to convert 7F3 hexadecimal to decimal:

1. The hexadecimal number has three digits, so start with the section of Figure E-1 labeled *Third Digit*. Find *7* in the column labeled *Hex;* the corresponding entry in the column labeled *Dec* is *1792*.
2. In the section of the table labeled *Second Digit,* find *F;* its decimal equivalent is *240*.
3. Use the section labeled *First Digit* for the rightmost digit; the decimal equivalent of *3* hexadecimal in this digit position is *3*.

4. Add the decimal equivalents of each digit to find the decimal equivalent of the number: 1792 plus 240 plus 3 equals 2035.

Figure E-2 shows a few more examples of converting a hexadecimal number to decimal using the columns in Figure E-1:

Hexadecimal Number	*Fourth Digit*		*Third Digit*		*Second Digit*		*First Digit*		*Decimal Equivalent*
9								=	9
D							13	=	13
1B					16	+	11	=	27
64					96	+	4	=	100
100			256	+	0	+	0	=	256
C5A			3072	+	80	+	10	=	3162
1000	4096	+	0	+	0	+	0	=	4096
FFFF	61,440	+	3840	+	240	+	15	=	65,535

Figure E-2. *Converting a hexadecimal number to decimal.*

APPENDIX

F

A Sample Menu System

Figures F-1 through F-9 show the BAT and DOC files required for a menu system with four sub-menus. Although the program and directory names might be different from yours, seeing the menus and corresponding batch files might help you put your menu system together.

The Main Menu batch file (MAINMENU.BAT), shown in Figure F-5, assumes that an environment variable named DR contains the letter of the drive from which the menu system is to be run and that the application programs are on drive C.

The Main Menu (MAINMENU.DOC) and sample word-processing sub-menu (WORDMENU.DOC), shown in Figures F-1 and F-3, include ANSI.SYS commands that set various foreground colors for the title frame and menu selections. Each WORDMENU.DOC choice is cyan, which matches the color of the word-processing selection in MAINMENU.DOC.

❑ **NOTE:** *The [E] symbol in the following figures represents the Escape character (ASCII code 27). To enter an Escape character in Edlin, press Ctrl-V, then a left bracket ([).*

```
1: [E][31m
2: [E][1m[E][33m F O R S Y T H - W O L F  C O M M U N I C A T I O N S [E][31m[E][0;31m
3:
4:
5:                    [E][1;37mWriting and Business Management System
6:
7:
8:
9: [E][1;32m
10: [E][1;32m  F1   Blank the screen
11: [E][1;32m
12: [E][0;33m
13: [E][0;33m  F3   123 Menu
14: [E][0;33m
15: [E][0;36m
16: [E][0;36m  F5   Word Processing Menu
17: [E][0;36m
18: [E][33m
19: [E][33m  F7   Telecommunications Menu
20: [E][33m
21: [E][1;36m
22: [E][1;36m  F9   Printer Setup Menu
23: [E][1;36m

[E][0;37m
[E][0;37m  F2   Back up the fixed disk
[E][0;37m
[E][1;34m
[E][1;34m  F4   Windows
[E][1;34m
[E][1;35m
[E][1;35m  F6   RBase
[E][1;35m
[E][1;31m
[E][1;31m  F8   MaxThink
[E][1;31m
   [E][0;32m
   [E][0;32m  F10  Return to DOS
   [E][0;32m  [E][m
```

Figure F-1. *The sample Main Menu (MAINMENU.DOC).*

```
1:
2:  E[1m  F O R S Y T H - W O L F  C O M M U N I C A T I O N S  E[m||
3:
4:
5:                  E[1mSpreadsheet Directories                  E[m||
6:
7:
8:
9:
10:  F1  Budgets                          F2  Forecasts
11:
12:
13:  F3  Taxes                            F4  Proposals
14:
15:
16:  F5  Miscellaneous
17:
18:
19:
20:
21:
22:                                       F10  Return to Main Menu
23:
```

Figure F-2. *The sample Spreadsheet Menu (123MENU.DOC).*

```
1:  E[32m
2:  E[1m  E[33mF O R S Y T H - W O L F  C O M M U N I C A T I O N S E[32m  E[0;32m||
3:
4:
5:                  E[1;37mWord Processing Directories                  E[0;32m||
6:
7:                                                                      E[36m
8:
9:
10:  F1  Supercharging DOS                F2  Forsyth-Wolf
11:
12:
13:  F3  Personal                         F4  Batch files
14:
15:
16:  F5  PC-VT                            F6  Miscellaneous
17:
18:
19:
20:
21:
22:                                       F10  Return to Main Menu
23:                                       E[m
```

Figure F-3. *The sample word-processing sub-menu (WORDMENU.DOC).*

```
 1:
 2:  E[1m ▓▓▓▓▓▓▒▒▒▒  F O R S Y T H - W O L F  C O M M U N I C A T I O N S  ▓▓▓▓▓▓▒▒▒▒ E[m||
 3:
 4:
 5:                          E[1mTelecommunications Programs                          E[m||
 6:
 7:
 8:
 9:
10:
11:   F1   PC-Talk                                F2   PC-VT
12:
13:
14:
15:
16:
17:
18:
19:
20:
21:
22:                                              F10  Return to Main Menu
23:
```

Figure F-4. *The sample Telecommunications Menu (TELEMENU.DOC).*

```
 1:     @echo off
 2:     echo E[m
 3:     prompt
 4:  :START
 5:     cls
 6:     nocurs
 7:     %dr%
 8:     cd \menu
 9:     type mainmenu.doc
10:  :GET_RPLY
11:     reply
12:     if errorlevel 69 goto GET_RPLY
13:     if errorlevel 68 goto F10
14:     if errorlevel 67 goto F9
15:     if errorlevel 66 goto F8
16:     if errorlevel 65 goto F7
17:     if errorlevel 64 goto F6
18:     if errorlevel 63 goto F5
19:     if errorlevel 62 goto F4
20:     if errorlevel 61 goto F3
```

Figure F-5. *The sample Main Menu batch program (MAINMENU.BAT).* *(continued)*

Figure F-5. *continued*

```
21:     if errorlevel 60 goto F2
22:     if errorlevel 59 goto F1
23:     goto GET_RPLY
24:  :F10
25:     cls
26:     goto END
27:  :F9
28:     cls
29:     prompt
30:     call prntmenu mainmenu
31:     goto START
32:  :F8
33:     c:
34:     cd \max
35:     max
36:     goto START
37:  :F7
38:     cls
39:     type telemenu.doc
40:  :RPLY_COM
41:     c:
42:     reply
43:     if errorlevel 68 goto START
44:     if errorlevel 60 if not errorlevel 61 goto F2_COMM
45:     if errorlevel 59 if not errorlevel 60 goto F1_COMM
46:     goto RPLY_COM
47:  :F1_COMM
48:     cd \pc-talk
49:     pc-talk
50:     goto F7
51:  :F2_COMM
52:     cd \pc-vt
53:     pc-vt
54:     goto F7
55:  :F6
56:     cls
57:     cd \rbase
58:     rbase
59:     goto START
60:  :F5
61:     cls
62:     type wordmenu.doc
63:  :RPLY_WRD
64:     reply
65:     if errorlevel 69 goto RPLY_WRD
```

(continued)

Figure F-5. *continued*

```
66:     if errorlevel 68 goto START
67:     if errorlevel 65 if not errorlevel 68 goto RPLY_WRD
68:     c:
69:     if errorlevel 64 if not errorlevel 65 cd \word\misc
70:     if errorlevel 63 if not errorlevel 64 cd \pc-vt
71:     if errorlevel 62 if not errorlevel 63 cd \batch
72:     if errorlevel 61 if not errorlevel 62 cd \word\pers
73:     if errorlevel 60 if not errorlevel 61 cd \word\for-wolf
74:     if errorlevel 59 if not errorlevel 60 cd \word\super
75:     if errorlevel 0 if not errorlevel 59 goto RPLY_WRD
76:     word
77:     %dr%
78:     cd \menu
79:     goto F5
80:   :F4
81:     c:
82:     cd \windows
83:     win
84:     goto START
85:   :F3
86:     cls
87:     type 123menu.doc
88:   :RPLY_123
89:     reply
90:     if errorlevel 69 goto RPLY_123
91:     if errorlevel 68 goto START
92:     if errorlevel 64 if not errorlevel 68 goto RPLY_123
93:     c:
94:     if errorlevel 63 if not errorlevel 64 cd \123\misc
95:     if errorlevel 62 if not errorlevel 63 cd \123\proposal
96:     if errorlevel 61 if not errorlevel 62 cd \123\taxes
97:     if errorlevel 60 if not errorlevel 61 cd \123\forecast
98:     if errorlevel 59 if not errorlevel 60 cd \123\budget
99:     if errorlevel 0 if not errorlevel 59 goto RPLY_123
100:    123
101:    %dr%
102:    cd \menu
103:    goto F3
104:  :F2
105:    call bakup
106:    goto START
107:  :F1
108:    cls
109:    reply
110:    goto START
```

(continued)

Figure F-5. *continued*

```
111:  :END
112:     %dr%
113:     normcurs
```

```
1:
2:  E[1m  F O R S Y T H - W O L F  C O M M U N I C A T I O N S  E[m
3:
4:
5:                    E[1mPrinter Setup Menu                    E[m
6:
7:
8:
9:
10:  F1                F2
11:
12:
13:  F3                F4
14:
15:
16:  F5                F6
17:
18:
19:  F7                F8
20:
21:  F9  Reset         F10  Return to
22:      Printer           DOS
23:
```

```
► Press F1-F9 to send a
  command to the printer.

► Press the right arrow key
  to display more commands.

► Press F10 when finished.
```

Figure F-6. *The Printer Setup Menu (PRNTMENU.DOC).*

```
1:  E[14;49HE[1m Typeface Commands     E[m
2:  E[9;8H           E[9;24H
3:  E[10;8HExpanded  E[10;24HCompressed
4:  E[12;8H          E[12;24H
5:  E[13;8HElite     E[13;24HProportional
6:  E[15;8H          E[15;24H
7:  E[16;8HBold      E[16;24HItalic
8:  E[18;8H          E[18;24H
9:  E[19;8HUnderline E[19;24HNLQ
```

Figure F-7. *The Typeface commands (PRNTOPT1.DOC).*

```
1:  [14;49H[1mPage Format Commands[m
2:  [9;8H              [9;24H
3:  [10;8H1½ space     [10;24HDouble space
4:  [12;8HTriple       [12;24H8 lines
5:  [13;8HSpace        [13;24HPer inch
6:  [15;8HTiny         [15;24H
7:  [16;8HPrint        [16;24H11-inch page
8:  [18;8HAddress      [18;24HDisable
9:  [19;8Hlabels       [19;24Hpaper out
```

Figure F-8. *The Page Format commands (PRNTOPT2.DOC).*

```
 1:     @echo off
 2:     cls
 3:     type prntmenu.doc
 4:     if "%1"=="mainmenu" echo [22;24HMain Menu
 5:  :CHOICE_1
 6:     type prntopt1.doc
 7:  :REPLY_1
 8:     reply
 9:     if errorlevel 77 if not errorlevel 78 goto CHOICE_2
10:     if errorlevel 69 goto REPLY_1
11:     if errorlevel 69 goto END
12:     if errorlevel 67 echo [@> prn
13:     if errorlevel 66 if not errorlevel 67 echo [n> prn
14:     if errorlevel 65 if not errorlevel 66 echo [-1> prn
15:     if errorlevel 64 if not errorlevel 65 echo [4> prn
16:     if errorlevel 63 if not errorlevel 64 echo [E> prn
17:     if errorlevel 62 if not errorlevel 63 echo [p1> prn
18:     if errorlevel 61 if not errorlevel 62 echo [M> prn
19:     if errorlevel 60 if not errorlevel 61 echo <Alt-15>> prn
20:     if errorlevel 59 if not errorlevel 60 echo [W1> prn
21:     goto REPLY_1
22:  :CHOICE_2
23:     type prntopt2.doc
24:  :REPLY_2
25:     reply
26:     if errorlevel 77 if not errorlevel 78 goto CHOICE_1
27:     if errorlevel 69 goto REPLY_2
28:     if errorlevel 68 goto END
29:     if errorlevel 67 echo [@> prn
30:     if errorlevel 66 if not errorlevel 67 echo [8> prn
31:     if errorlevel 65 if not errorlevel 66 echo [C0<Alt-7>> prn
32:     if errorlevel 64 if not errorlevel 65 echo [C<Alt-128>[N<Alt-6>> prn
```

Figure F-9. *The Printer Setup Menu batch file (PRNTMENU.BAT).* *(continued)*

Figure F-9. *continued*

```
33:     if errorlevel 63 if not errorlevel 64 echo <Alt-15>[ESO[E3<Alt-15>> prn
34:     if errorlevel 62 if not errorlevel 63 echo [E0> prn
35:     if errorlevel 61 if not errorlevel 62 echo [E31> prn
36:     if errorlevel 60 if not errorlevel 61 echo [E3H> prn
37:     if errorlevel 59 if not errorlevel 60 echo [E36> prn
38:     goto REPLY_2
39:  :END
40:     cls
```

Glossary

A

ANSI.SYS: A program called a *device driver* that tells DOS how to manage the display and keyboard. *ANSI* stands for *American National Standards Institute. See also* ANSI.SYS command.

ANSI.SYS command: A command recognized by ANSI.SYS that controls the cursor, display attributes, and keyboard assignments. Each ANSI.SYS command begins with an Escape character (character 27) followed by a left bracket. If you want to use ANSI.SYS commands, the ANSI.SYS file must be on your system disk and the CONFIG.SYS file must be in the root directory of the system disk and must contain the configuration command *device=ansi.sys.*

ASCII: The code that most computers use to represent letters, numbers, and symbols. *ASCII* stands for *American Standard Code for Information Interchange.*

Attribute: *See* Display attribute.

B

Batch command: The name of a batch file without the extension BAT. When you type a batch command at the DOS prompt, DOS carries out the commands in the batch file.

Batch file: A text file with the extension BAT that contains DOS commands. When you type the name of the batch file, DOS carries out the commands in the file.

Bell: The sound, usually a beep, that a computer or printer makes when it receives ASCII character 7 (which can be typed by holding down the Alt key, then pressing the 7 key on the numeric keypad).

Binary: The base-2 numbering system, whose only digits are 0 and 1. Computers use the binary system because the digits can be represented by the presence (1) or absence (0) of voltage.

Bit: The smallest quantity that a computer can measure or detect; corresponds to a binary digit (either 0 or 1). A byte is made up of 8 bits.

Box-drawing characters: The characters that consist of corners, horizontal lines, vertical lines, and intersections, used to draw lines and boxes. The IBM extended character set includes three sets of such characters; their codes range from 179 through 218.

Buffer: An area of memory that DOS uses as an intermediate storage area when it reads from or writes to a disk. You can control the number of buffers with the Buffers configuration command.

Byte: The unit of measure for computer memory and disk storage. Each byte contains 8 bits and can store one character (a letter, number, or symbol).

C

Cluster: DOS stores files on disk in clusters, each cluster being a certain number of sectors. On a floppy disk, depending on the format, one cluster equals one or two sectors, with a sector equalling 512 bytes. On a hard drive, a cluster may consist of several sectors. This is why even the most minute of files will occupy up to 2K of disk space; the file exists in a cluster on disk, not a sector.

Command: An instruction you give to a program such as DOS, ANSI.SYS, Debug, Edlin, or an application program.

Command path: The list of path names that tells DOS where to look for command files that aren't in the current directory.

COMSPEC: A configuration command that tells DOS where to find the program that interprets what is typed in response to the system prompt. Except in unusual circumstances, the program is COMMAND.COM. *COMSPEC* stands for *command specification.*

CONFIG.SYS: A file of configuration commands that DOS reads each time it starts.

Configuration: The makeup of a computer system, as described to DOS in the CONFIG.SYS file. The configuration includes devices that require special programs, such as a mouse; memory set aside for special uses, such as file buffers; and some aspects of system operation, such as how often to check whether Ctrl-C has been pressed.

Configuration command: An instruction in the CONFIG.SYS file that tells DOS some detail of the system configuration, such as the name of a device driver (*device=ansi.sys*) or the number of buffers to use for disk operations (*buffers=20*).

Control character: One of the first 32 characters (codes 0 through 31) that, in the ASCII code, represent an action—such as tab or backspace—rather than a character.

Conventional Memory: That memory which exists from 0K through 640K. Conventional memory is where DOS and where DOS programs normally run.

D

Debug: A DOS program that lets you examine memory, alter memory, load sectors of data from a disk, and create assembly language programs.

Device driver: A program that tells DOS how to operate, or *drive,* a device. It can be a separate program with the extension SYS, such as ANSI.SYS, or part of another program (many application programs include device drivers for printers, for example). A device driver might give instructions for a device that DOS considers uncommon (such as MOUSE.SYS), or it might give instructions for handling a common device in an uncommon way (such as ANSI.SYS). *See also* Configuration command.

Display attribute: A feature of the display that can be controlled, such as high intensity, blinking, or color. One way to control display attributes is with ANSI.SYS commands.

E

Environment: An area of memory in which DOS stores definitions of systemwide features such as the command path or system prompt.

Environment variable: A description that is stored in the environment. It consists of a name and a value separated by an equal sign, such as *PROMPT=[$p].* In addition to the environment variables created by DOS and other programs, you can create your own with the Set command.

Errorlevel: A numeric value set by some programs that you can test with the ERRORLEVEL option of the If batch command.

Escape character: An ASCII control character—code 27—often used to mark the beginning of a series of characters that represent a command rather than data. So called because it escapes from the usual meaning of the ASCII code and allows commands to be interspersed in a file of data, especially for data transmission.

Escape sequence: A series of characters, usually beginning with the Escape character, that is to be interpreted as a command and not as data.

Expanded memory: Additional memory that can be installed on a PC. To use expanded memory, you need a special expanded memory manager program: Programs must be written specifically to use such memory. Use of expanded memory is governed by the Lotus-Intel-Microsoft (LIM) specification. Up to 32MB of expanded memory can be installed.

Extended character set: The characters assigned to codes 128 through 255 on IBM and IBM-compatible computers. These characters are not defined by the ASCII standard.

Extended key code: The two-number code that represents pressing a key outside the typewriter portion of the keyboard, such as a function key, cursor-control key, or combination of the Ctrl or Alt key and another key. The first number is always 0, and is separated from the second number by a semicolon. *See also* Key code.

Extended memory: Additional memory, above the 1MB mark, installed on a PC with an 80286 or later microprocessor. This memory cannot be used directly by DOS or DOS applications. However, using an Extended Memory Specification (XMS) memory manager, DOS can use parts of extended memory and convert some or all of it into expanded memory for programs that require it.

F

FCB (File Control Block): An area of memory used by DOS to describe the name, size, location, and other information about a file that is being used by a program.

Form feed: The character that causes a printer to advance to the top of the next page (or form). When sent to the display, it clears the screen by scrolling all the lines up to the top of the next screen (page), which is blank. Its character code is 12.

H

Hexadecimal: The base-16 numbering system, whose digits are 0 through 9 and A through F (the letters A through F represent the decimal numbers 10 through 15). The system is often used in computer programming because hexadecimal is a convenient shorthand for working with binary, the base-2 numbering system used by computers.

High Memory Area (HMA): The first 64K of extended memory. The HMA exists on 286 or later PCs with at least 256K of extended memory. The HIMEM.SYS device driver establishes control over this area, which allows DOS and its applications access to the extra 64K. In DOS 5, the HMA can be used to store DOS's own programs, increasing available conventional memory.

K

Key code: The number that represents pressing a key in the typewriter portion of the keyboard (a letter, number, or punctuation mark). The key code is the ASCII code for the corresponding character. *See also* Extended key code.

M

Macro: A miniprogram, or group or commands or instructions, represented by a single word. *Macro* is short for *macroinstruction.* Typically, a macro is a word or command that causes a longer command or series of commands to be executed. In DOS 5, the Doskey command lets you create one-word macros that represent longer, complex DOS commands.

N

Non-keyboard character: A character that cannot be typed on the keyboard by pressing a single key. In general, a character whose code is less than 32 and greater than 126.

P

Path: The list of directory names that defines the location of a directory.

Printable character: Sometimes used to describe a character from 32 through 127, the ASCII codes for characters to be printed, as opposed to characters 0 through 31, the ASCII codes for control characters.

Printer command: A series of characters that cause a printer to take some action, such as changing the typeface or line spacing, rather than printing a character. Most printer commands are *escape sequences.*

R

RAM: Short for *random access memory,* the memory that a computer uses for programs and data. The contents of RAM change often and are lost when the computer is turned off.

RAM disk: An area of memory that behaves like a disk drive under the control of a device driver such as VDISK.SYS or RAMDRIVE.SYS.

RAMDRIVE.SYS: A device driver that tells DOS to treat an area of memory as a disk drive.

Redirection symbol: The greater-than (>), less-than (<), and vertical bar (¦) symbols. They are used to redirect input or output and to pipe the output of one program to another.

Register: An area of memory on the microprocessor chip used to hold 1-byte or 2-byte values.

ROM: Short for *read-only memory.* A type of memory whose content is permanently recorded and not lost when the computer is turned off. This type of memory is normally used to store a program or data that never changes, such as the shape of the characters used on the display.

S

Script file: A text file that contains a complete sequence of commands, or a *script,* for a program so that input can be redirected from the console to the file. It is used to automate the operation of application programs such as Debug.

Shell: A program that shows itself to the person using the computer, then passes commands to a different program to be carried out. It's called a *shell* because it surrounds the other program, hiding it from view. Version 4 of DOS introduced such a shell program, which lets you use many DOS commands without having to type commands or file names. Many other DOS shell programs are available.

U

Upper memory area: The area of memory between 640K and 1MB in a DOS computer. Also called *reserved memory* or *high DOS memory,* the upper memory area is traditionally the place where the BIOS and ROM instructions are stored for various parts of the computer.

Upper memory block: An unused portion of the upper memory area. In DOS 5, an upper memory block can be filled with RAM on a 386 or later PC using the EMM386.EXE device driver. The Devicehigh and Loadhigh commands can then be used to transfer device drivers and memory-resident software into an upper memory block, increasing the total amount of available conventional memory.

V

VDISK.SYS: A device driver that tells DOS to treat an area of memory as a disk drive.

Virtual disk: *See* RAM disk.

I N D E X

Italic page numbers refer to figures.

Special Characters

$ (Doskey command character) 274–76
$ (Prompt command character) 24–*25*
% (batch replaceable parameter symbol) 91, 105
% (environment variable symbol) 146
- (Debug command prompt) 60
.. (parent directory symbol) 291–92
< (redirection operator) 77
> (redirection operator) 43
>> (redirection operator) 104
@ (batch display-suppression operator) 85, 88, 90
^ (Control-key symbol) 5
¦ (pipe operator) 87, 88, 103
386 enhanced mode (Windows) 372
386MAX software 184
8086/8088-based computers 182–83
80286-based computers 171–73, 182–83
80386-based computers 173–82, 184

A

a (Debug Assemble command) 79–80
access, file/directory. *See* file/directory attributes
access, speeding up disk. *See* disk buffers; disk caches; Fastopen command; RAM disks
action keys 13–14
ADD.BAT file 298–301
addresses, memory 61–62
aliases, directory 292–95
Alternate-key (Alt) sequences
  ASCII character entry 12–14
  Doskey usage *269*, 270, 273
ANSI.SYS device driver
  batch file screen design and 85
  command overview 22–24
  command quick reference 402–7
  commands in system prompts 24–25
ANSI.SYS device driver, *continued*
  display-control commands 26–31, 106–8, 344–45
  installation 20–21
  keyboard-control commands 35–39, 276–81
  key codes 422–*25*
antistatic protection 392–93
application programs and colors 221
arrow keys. *See* direction keys
ASCII character set
  control characters 5–7, 13, 15, *419*
  entering non-keyboard characters 12–14
  extended character set 7–8, 12–13, *417* (*see also* box-drawing characters)
  extended key codes for *424–25*
  null character 15
  overview 4–5, 58
  standard text character set 7, *416*
  tables *416–19*
ASK.COM program 127–32
Assemble command (Debug) 79–80
assembly language 79
AT computer, IBM 281–83
Attribute command (attrib) 87, 134–36, 291
attributes. *See* display attributes; file/directory attributes
AUTOEXEC.BAT files
  emergency boot disks and 355
  interactive 130–32
  multiple 378–84
  standard colors in 219
  startup disks and 181–82

B

backfill memory 183
background colors 218–19
backing up files/directories
  Backup and Restore commands 317, 327, 359

backing up files/directories, *continued*
batch files and Xcopy command 317–26
overview 316–17
RAM disks 189, 199
strategies 316–17, 323–24, 326–27
third-party software 317
Backspace key 6, 14, 262
Backup command 317, 327
Bad command or file name message 88
BAKUP.BAT file 317–27
bank-switched memory 163
batch display-suppression operator (@) 85, 88, 90
batch files
backing up with 317–27
calling other 94–102
calling self 96–97, 101–2
command-line parameters 90–91
creating, with Doskey 270–71
creating fixed-format memo files with 303–7
creating note files with 301–3
creating RAM disks with 189–99
creating text files with 295–301
cursor control with 210–15
date-stamping files with 103–6, 307–8
designing and debugging 84–89
display control with 106–8, 120–25, 215–21
displaying blank lines with 90
displaying help with 118–19
display suppression with 85, 88, 90, 91–92
Doskey macros vs. 272
DOS startup with (*see* AUTOEXEC.BAT files)
environment variables in 146–48
finding files on hard drives with 308–11
formatting disks with 311–14
interactive 112–18, 130–32
managing multiple configurations with 378–84
menu systems (*see* menu systems)
moving between directories with 292–95
overview *ix–x*
printer control with 92
printer setup with 241–58, 338–40
batch files, *continued*
on RAM disks 188–89
redirecting output of 102–3
restoring current directory with 314–16
restoring system prompt with 25
security systems 125–26, 132–34, 136–38, 216–17
sound-generation with 92–94
subdirectories for 291
terminating 126–27
BATCH subdirectory 291
beeps and bells 7, 86
BIGCURS.COM program 211–12
binary numbers 8
BIOS (Basic Input/Output System) 162
BLANK.BAT file 215–16
BLANKSEC.BAT file 216–17
blinking characters 29–30
bold characters 29–30
BOOT.BAT file 383–84
boot disks 354–57
BOOT-DOS.BAT file 380–82
boot files. *See* AUTOEXEC.BAT files; CONFIG.SYS configuration files
booting DOS. *See also* AUTOEXEC.BAT files
from keyboard 21
with multiple configurations 378–84
with programs 190–91
boot monitor programs 384
BOOT-WIN.BAT file 380–82
BOX.BAT file 278–81
box-drawing characters
keyboard layout for 277–81
overview 8
printing 49
screen design with 16–18, 85
table *418*
Break command 86–87, 154. *See also* Ctrl-Break keys
Break configuration command 154. *See also* Ctrl-Break keys
Break key 6
breaks. *See* Ctrl-Break keys
buffers, disk 151–53. *See also* disk caches; Fastopen command
Buffers configuration command 152–53

bugs 58
bulletproofing 85
bytes 4, 58

C

caches, disk. *See* disk caches
Call batch command 94–97
calling batch files 94–102
caret (^) as Control-key symbol 5
carriage-return character 63, 103. *See also* Enter key
chaining batch files 94
Change Directory command (cd) 292, 314
characters. *See also* keyboards; keys
  ASCII (*see* ASCII character set)
  blinking and bold 29–30
  box-drawing (*see* box-drawing characters)
  carriage-return 63, 103
  compressed 43–44, 236–39
  DOS editing 260–64
  Doskey command-separator 267–68
  Doskey editing 265–67
  Doskey special 274–76
  Escape (*see* Escape character)
  filling screen with 74–79
  foreign language 285–88
  form-feed 6, 92
  Greek 8
  line-feed 63–64
  non-keyboard 12–13
  normal 29–30
  null 15
  printing extended 49–54
  printing small 236–39
  Prompt command *25, 207*
Check Disk command (chkdsk) 100, 309–11
clock, system. *See* date; time
code pages 285–88
color monitors 29–30, 33–34. *See also* EGA displays; VGA displays
colors
  application programs and 221
  screens 26–*27*, 217–20, 344
  system prompts 209–10
comfort, convenience, and maintenance issues 395–97
COMMAND.COM files
  accidental deletion of 357–59
  environment variables for 144
  loading multiple copies of 97–103
  protecting 87
  as shell programs 154–55
Command command 97–103
command history. *See* Doskey command
command-line parameters *xii*, 90–91
command macros. *See* Doskey command
command processors. *See* COMMAND.COM files
command prompt, Debug 60
command prompt, DOS. *See* prompts, system
commands
  assigning, to keys 37–39, 277
  batch (*see* batch files)
  configuration (*see* CONFIG.SYS configuration files)
  display-control and keyboard-control (*see* ANSI.SYS device driver)
  editing DOS 260–68
  filter commands 103, 105–6, 168–69
  history and macros (*see* Doskey command)
  parameters *xii*, 90–91
  printer (*see* printer commands)
  redirecting input/output of (*see* input/output redirection)
  search paths 89, 144–45, 147–48
  suppressing display of 85, 88, 90, 91–92
  timing of 105–6
command-separator character (Doskey) 267–68
compatibility
  DOS versions 151, 356, 385–87
  memory 160–61
  printers 51–53
compressed print 43–44, 236–39
computers
  comfort, convenience, and maintenance issues 391–97
  memory expansion strategies 182–84, 390–91

computers, *continued*
memory optimization for 80286-based 170–73
memory optimization for 80386-based 173–82
memory usage and microprocessors 160–61
operating systems (*see* DOS; extended DOS environments; Windows)
overcoming speed and capacity limitations 390–91
problems (*see* problems, computer)
speeding up IBM PC/AT and PS/2 keyboards 281–83
suggested buffers for different *153*
third-party memory managers 182–84
COMSPEC environment variable 144
CON device name 204
config juggler programs 384
CONFIG.SYS configuration files
checking for breaks 154
creating disk buffers 151–53
creating file handles or file control blocks 153
emergency boot disks and 355
installing ANSI.SYS 20–21
installing device drivers 150–51
installing disk caches 200–201
installing memory managers 155–58
installing RAM disks 187–99
multiple 378–84
sample, for third-party software 183–84
startup disks for modifying 177–80
console device 204. *See also* keyboards; screens
control characters 5–7, 13, 15, *419*
Control-key (Ctrl) sequences
ASCII character entry 13–14
ASCII control characters 5–7
breaks (*see* Ctrl-Break keys)
Doskey usage 265–68
restarting DOS 21
switching keyboard layouts 287
Control-key symbol (^) 5
conventional memory
backfilling 183
converting video memory to 184
conventional memory, *continued*
examining, with Check Disk command 100
examining, with Debug command (*see* Debug command)
examining, with Mem command 166–70
Fastopen buffers 202
maximizing, for 80286-based computers 170–73
maximizing, for 80386-based computers 173–82
overview 162
Copy command
combining files 314–16
creating files 59
printing files 51
sending printer commands 45
suppressing messages of 91–92
country codes *287*
COUNTRY.SYS device driver 286
Ctrl-Break keys
checking for 86–87, 154
entering 13–14
terminating batch files 126–27
Ctrl key. *See* Control-key (Ctrl) sequences
cursor control
with ANSI.SYS commands 28–30, 107, *403–4*
in menus 345
with programs 210–15
with Prompt commands 207
cursor-movement keys. *See* direction keys

D

d (Debug Dump command) 660–65
date
stamping files with 103–6, 307–8
in system prompts 205–9
Date command 103
Debug command
assembling programs 79–80
changing length of files 70–74
creating programs with script files 74–80
dumping files 60–65
editing files 66–67, 70–71
overview 58–60

Debug command, *continued*
quitting 69
registers and 71–74
saving and naming files 67–69
unassembling programs 80
verifying work 69–70
Debug command prompt (-) 60
debugging batch files 87–88
decimal numbers, converting hexadecimal numbers to 428–29
Delete key (Del) 263
delete tracking 363–65
designing batch files 84–89
DESQview environment 378
Device configuration command 20–21, 150–51
device drivers
DOS versions and 356
emergency boot disks and 356–57
expanding 180–81
installing 20–21, 150–51
international support 285–86
memory managers 155–58
moving, into upper memory 158, 176–77
third-party 164
Windows vs. DOS 376–78
Devicehigh configuration command 157–58, 177
device names
console 17, 204
null 91–92
printer 14, 43–44
dip switches 53
direction keys
DOS command-line editing usage 260–64
Doskey usage 265–70
extended key codes for *423*
directories
access (*see* file/directory attributes)
aliases 292–95
backing up (*see* backing up files/directories)
caching 202
in command search paths 147–48
current, in system prompt 205–9
displaying subdirectories only 290
moving between 291–92
directories, *continued*
parent 291–92
RAM disk entries for 189, 199
restoring current 314–16
structures 290–91
disk access speedup. *See* disk buffers; disk caches; Fastopen command; RAM disks
DISKBAK.BAT files 317–27
disk buffers 151–53. *See also* disk caches; Fastopen command
disk caches. *See also* disk buffers; Fastopen command
advantages and disadvantages 199–200
installing 200–201
secondary caches 151, 152
Windows 373–74
disk drive letters
setting highest 198
in system prompts 205–9
diskettes. *See* floppy disks
DISKFORM.BAT file 311–14
disks. *See also* floppy disks; hard drives
backing up (*see* backing up files/directories)
caches (*see* disk caches)
densities 313
directories (*see* directories)
files (*see* files)
formatting (*see* formatting disks)
RAM (*see* RAM disks)
recovering files from damaged 359–65
unformatting 366–69
display attributes. *See also* colors
ANSI.SYS commands to change *405–6*
setting 26–31
setting, with batch files 106–8, 215–21
displays. *See* color monitors; EGA displays; screens; VGA displays
dollar-sign ($) command characters (Doskey) 274–76
dollar-sign ($) command characters (Prompt) *24–25*
DOS
batch files (*see* batch files)
command history and macros (*see* Doskey command)

DOS, *continued*
command-line editing 260–64
commands (*see* commands)
configuration (*see* CONFIG.SYS configuration files)
disk management (*see* disks)
environment (*see* environment, DOS)
extended environments 378
files (*see* files)
foreign language support 285–88
memory management (*see* memory; memory, optimizing; memory managers)
menu systems (*see* menu systems)
moving, into high memory 156–57, 170–82
organization of this book about *x, xi, xiii–xiv*
programs (*see* programs)
prompt (*see* prompts, system)
recovery utilities 358–59
restarting, from keyboard 21
restarting, with program 190–91
shell (*see* DOSShell command)
startup disks for modifying 177–80
subdirectories for 291
versions (*see* versions, DOS)
Windows and 376–78
DOS configuration command 156–57, 172–73
Doskey command
assigning menus to keys 346
command history 268–71
command-line editing 264–68
creating batch files 270–71
creating directory aliases 294–95
creating macros 272–76
editing keys 265–67
multiple commands on one line 267–68
in upper memory 11
DOSShell command 154, 181, 350–*51*
DOS subdirectory 291
dot-matrix printers. *See* Epson-compatible printers
double period (..) as parent directory symbol 291–92
drive letters. *See* disk drive letters
Dump command (Debug) 60–65

E

e (Debug Enter command) 66–67
Echo batch command
creating beeps 92–93
displaying blank lines 85, 90
sending console commands 24
sending printer commands 43–45, 92
setting standard colors 219
suppressing messages 85, 88, 90
Type command vs. 247
editing DOS commands
at DOS command line 260–64
with Doskey 264–68
editing files. *See* Debug command
Edlin command 14
EGA displays
setting display lines 204–5
Windows and 376
EGA.SYS device driver 376
electrical problems 392–96
emergency boot disks 354–57
EMM386.EXE device driver and command 157–58, 174–75
EMM386.SYS device driver (Windows) 376
EMS (expanded memory specification) 162–63
enhanced keyboards 21, 283–85
enhanced mode (Windows) 372
ENV.BAT 146–47
environment, DOS
command processors and 99–100
increasing size of 154–55
memory usage by 149–50
overview 144–45
system prompt and 208
variables 144–45, 148–49
environments, extended-DOS 378
environment variables
creating 148–49
default 144–45
setting and deleting 145
Enter command (Debug) 66–67
Enter key 6, 13, 14, 37. *See also* carriage-return character
Epson-compatible printers
page-format commands *411–12*

Epson-compatible printers, *continued*
printer-control commands *412*
printer setup for *248–49*
printing extended character set 49–54
printing small characters 236–38
resetting, with program 240–41
sending commands to 43–46
typeface commands *411*
Erase commands (ANSI.SYS) 404
errorlevels 112
error messages. *See* messages
errors, batch file 87–88
Escape character
ANSI commands as escape sequences 22
Control-key sequence for 6
entering, into text files 31–32, 119
printer commands with *46*
printer commands without *45*
Prompt command code for 23, 24, *25*
Exit command 98
expanded memory
backfilling conventional memory with 183
converting extended memory to 157–58, 174–75
disk caches in 200–201
examining 166–170, 175
Fastopen buffers in 202
managers 157–58, 174–75
overview 162–63
RAM disks in 187–88
Windows and 375–76
expanded memory specification (EMS) 162–63
expanded print 44–45
extended character set. *See also* box-drawing characters
Alternate-key method of entering 12–13
overview 7–8
table *417*
extended DOS environments 78
extended key codes *423–25*
extended memory. *See also* high memory area (HMA)
converting, to expanded memory 157–58, 174–75
disk caches in 200–201
examining 166–70
extended memory, *continued*
managers 151, 155–57, 171–72
overview 163
RAM disks in 187–88
Windows and 375–76
extended memory specification (XMS) 156, 164
extensions, file-name 290

F

FAST.COM program 282–83
Fastopen command 202
FCBS configuration command 153
File Allocation Tables (FAT) 360
FILE.BAT file 295–98
file control blocks (FCBs) 153
file/directory attributes
hidden 134–36
read-only 87, 291
File not found message 87
files. *See also* batch files; programs; text files
access, controlling (*see* file/directory attributes)
access, speeding up (*see* disk buffers; disk caches; Fastopen command; RAM disks)
backing up (*see* backing up files/directories)
buffers 151–53, 202
caching 202
configuration (*see* CONFIG.SYS configuration files)
date-stamping 103–6
delete tracking 363–65
directories of (*see* directories)
editing (*see* Debug command)
finding, on hard drives 308–11
naming 290
saving recovery information for 359–63
setting RAM disk maximum 189, 199
specifying number of open 153
swap 374–75
temporary 145
undeleting 369–71
zero-length 137–38
Files configuration command 153
filter commands 103, 105–6, 168–69

FINDFILE.BAT file 308–11
Find filter command 103, 105–6
fixed disks. *See* hard drives
fixed-format memo files 303–7
FLASH.COM program 74–79
floppy disks. *See also* disks
  emergency boot 354–57
  formatting (*see* formatting disks)
  protecting 391–92
  startup 177–80
For batch command
  displaying path directories 147–48
  using, outside of batch files 92
foreground colors 217–19
foreign language keyboards 285–88
Format command 177, 311–14, 354, 360, 365–66
format commands, printer. *See* page-format commands, printer
formatting disks
  with batch files 311–14
  emergency boot disks 354
  overview 360
  quick-formatting disks 365–66
  startup disk 177
form-feed characters 6, 92
full-screen editors 59
function keys
  DOS command-line editing usage 260–64
  Doskey usage 265–71
  extended key codes for *423*

G

General Failure error reading drive A message 396–97
Getting Started with MS-DOS 5 and the New *MS-DOS Shell* 350
GO.BAT file 292–95
graphics commands, printer 42
greater than signs (>, >>) as redirection operators 43, 104
Greek characters 8

H

hard drives. *See also* disks
  advantages 391
  backing up (*see* backing up files/directories)
  finding files on 308–11
hardware configuration 144
help text, batch file display of 118–19
Hewlett-Packard–compatible printers
  page-format commands *413–14*
  printer-control commands *414*
  printer setup for *249–50*
  printing extended character set 49–54
  printing small characters 238–39
  sending commands to 46–48
  typeface commands *413*
hexadecimal numbers 8–9, 58, 428–29
hidden file/directory attributes 134–38
high-DOS memory. *See* upper memory blocks (umbs)
high memory area (HMA). *See also* extended memory
  accessing 171–72
  examining 166–70
  moving DOS into 172–73
  overview 156, 163–64
HIMEM.SYS device driver 151, 155–57, 163–64, 171–72, 375
history, command. *See* Doskey command
hyphen (-) as Debug command prompt 60

I

IBM extended character set. *See* extended character set
IBM PC/AT and PS/2 keyboards 281–83
IBM printers 239–41. *See also* Epson-compatible printers
If batch command 112–13, 118
incompatibility. *See* compatibility
initializing disks. *See* formatting disks
input/output redirection
  of batch file output 102–3
  creating programs 75–79
  date-stamping files 103

input/ouput redirection, *continued*
errors 87
sending printer commands 43
Insert DOS disk in drive x and strike ENTER when ready message 357
Insert key (Ins) 262–63
Insufficient disk space message 326
Intermediate file error message 88
international characters 8, 13
international keyboard layouts 285–88
Invalid COMMAND.COM, insert COMMAND.COM disk in drive and strike any key when ready message 357
invisible cursors 214–15

K

Keyboard command (keyb) 181, 286–88
keyboards. *See also* keys
changing layout of 277–81
controlling, with ANSI commands 35–39, 402–7 (*see also* key codes)
copying printer commands from 45
enhanced 21, 283–85
foreign language 285–88
macros 35–39, 277, 345–46
non-keyboard characters 12–13
protecting 392
restarting DOS from 21
speeding up IBM PC/AT and PS/2 281–83
turning off Num Lock 283–85
KEYBOARD.SYS device driver 286
key codes
function and cursor-control keys *423*
standard ASCII characters *424–25*
keys. *See also* keyboards
Alternate-key sequences 12–14, *269*, 270, 273
assigning commands to 277
assigning menus to 345–46
Backspace 6, 14, 262
Break 6
command-editing 260–71
Control-key sequences (*see* Control-key (Ctrl) sequences)
keys, *continued*
Delete (Del) 263
Enter 6, 13, 14, 37
extended codes for *423–25*
Insert (Ins) 262–63
Tab 6, 14

L

Label not found message 88
labels, batch file 88
languages, assembly and machine 79
languages, foreign 285–88
laser printers. *See* Hewlett-Packard–compatible printers
Lastdrive configuration command 198–99
less than sign (<) as redirection operator 77
lighting 395
LIM EMS (expanded memory specification) 162–63
line-drawing characters. *See* box-drawing characters
line-editor programs 59
line-feed characters 63–64
Loadhigh (LH) command 157–58, 181–82, 265, 288
look-ahead buffers 152
Lotus 1-2-3 version 3 378
LPT1 printer port 43

M

machine language 79
macros
batch files vs. 272
Doskey (*see* Doskey command)
MAINMENU.BAT file 333–36, *434–37*
maintenance, computer 395–97
mathematical symbols 8
MEMO.BAT file 303–7
memo files 303–7
memory. *See also* conventional memory; expanded memory; extended memory; high memory area (HMA); upper memory blocks (umbs)
command processor usage 99

memory, *continued*
configuration commands 144, 155–58
disk caches (*see* disk caches)
displaying and changing contents of (*see* Debug command)
enabling extra 151
environment size 149–50, 154–55
examining, with Check Disk command 100
examining, with Mem command 166–70
expansion strategies 182–84, 390–91
file buffers usage 151–53
file/directory caching 202
file handles and control blocks usage 153
managers (*see* memory managers)
map 164–*65*
microprocessors and 160–61
overview 161–65
RAM disks (*see* RAM disks)
reconfiguring, with startup disks 177–80
video 184
Windows and 375–78
memory, optimizing
accessing high memory 171–72
accessing upper memory blocks 174–75
accessing video memory 184
disk caches (*see* disk caches)
moving device drivers into upper memory 158, 176–77
moving DOS into high memory 172–73
moving memory-resident programs into upper memory 158, 181–82
preparing DOS for upper memory blocks 156–57, 172–73
RAM disks (*see* RAM disks)
startup disk strategy 177–80
for Windows 373–75
Memory command (mem)
examining memory availability and usage 166–68
examining memory usage by programs 168–70
memory managers
DOS expanded 157–58, 174–75
DOS extended 151, 155–57, 163–64, 171–72
DOS 5 vs. Windows 376–77
third-party 183–84
Windows 375–76
memory-resident programs
interactive installation of 130–32
moving, into upper memory 158, 181–82
MENU.DOC file 16–18
MENU subdirectory 291
menu systems
accepting choices 115–18
adding colors 344
adding selections 337–38
assigning, to keys 345–46
batch files 333–37
complete sample 432–*39*
cursor control in 345
customizing 343–44
displaying help 118–22
DOSShell command 350–51
multiple sets of choices 122–25
overview 330–31
printer setup 241–58, 338–40
RAM disks and 346–50
required programs 331
subdirectories for 291
sub-menu selections 340–43
text screens 331–33
messages
Bad command or file name 88
File not found 87
General Failure error reading drive A 396–97
Insert DOS disk in drive x and strike ENTER when ready 357
Insufficient disk space 326
Intermediate file error 88
Invalid COMMAND.COM, insert COMMAND.COM disk in drive and strike any key when ready 357
Label not found 88
Non-system disk or disk error 357
Out of environment space 149–50, 208
suppressing 85, 88, 90, 91–92
Syntax error 88, 90
microprocessors and memory usage 160–61
Microsoft MS-DOS. *See* DOS
Microsoft Windows. *See* Windows
Mirror command
overview 359–60

Mirror command, *continued*
recovering from disk failures 362–63
recovering from file deletions 363–65
recovering from formatting 360–62
MIRROR.FIL file 360, 361
mnemonics 79
mode, protected 163
Mode command
controlling printers 43
setting display lines 204–5
speeding up IBM PC/AT and PS/2 keyboards 281–82
mode commands (ANSI.SYS) *405–6*
modes, Windows 372–73, 375–76
monitors. *See* color monitors; EGA displays; screens; VGA displays
monochrome monitors 29–30, 32–33
More filter command 168–69
MOUSE.SYS device driver 151
Move Cursor command (ANSI.SYS) 29
MOVE'EM software 183
MS-DOS. *See* DOS
multitasking 372

N

n (Debug Name command) 68
naming files 290
national language support 285–88
NOCURS.COM program 214–15
noises 92–94
non-keyboard characters 12–13
Non-system disk or disk error message 357
normal characters 29–30
NORMCURS.COM program 212–13
NOTE.BAT file 301–3
note files 301–3
NUL device name 85, 91–92, 137–38
null character 15
number systems
binary 8
hexadecimal 8–9, 58
hexadecimal-to-decimal conversion 428–29
Num Lock, turning off 283–85
NUMOFF.COM program 283–85

O

offsets, memory 62
operating systems. *See* DOS; extended DOS environments; Windows
optimizing memory. *See* memory, optimizing
optimizing Windows 373–75
Out of environment space message 149–50, 208
output. *See* input/output redirection; printers; screens

P

page eject command 6, 92
page-format commands, printer
Epson-compatible 411–12
HP-compatible 413–14
overview 42
on printer setup menu 242–*43,* 254–55
pages, printing files in 230–36
parallel printer ports 43
parameters, command-line *xii,* 90–91
parent directory (..) 291–92
partitions, hard disk 362–63
PARTNSAV.FIL file 360, 362
password systems 125–26, 132–34, 136–38
Path command 89
PATH environment variable 144, 145, 147–48
paths, command search 89, 144–45, 147–48
Pause batch command 127
PC. *See* computers
PC/AT keyboards 281–83
PCL command language. *See* Hewlett-Packard–compatible printers
PCTRACKR.DEL file 360, 363
percent sign (%) as symbol 91, 105, 146
periods (..) as parent directory symbol 291–92
PGM subdirectory 291
pipe operator (¦) 87, 88, 103
power-line conditioners 394–95
preventive maintenance 359–65
printable characters 7, 416
Print command 131–32, 181, 227–30
printer commands
categories and standards 42–43, 224–25, 410

printer commands, *continued*
Control-key sequences 6–7, 14
entering, into text files 46–48
Epson-compatible *46, 411–12*
HP-compatible 46–48, *413–14*
Reset command 54
sending, with Copy command 45
sending, with Echo command 43–45, 92
printer-control commands
Epson-compatible *412*
HP-compatible *414*
overview 42
printers. *See also* Epson-compatible printers; Hewlett-Packard–compatible printers
comfort, convenience, and maintenance issues 395–97
commands (*see* printer commands)
compatibility problems 51–53
device names 14, 43–44
operating 225–27
ports 43
printing extended character set 49–54
printing files in pages 230–36
printing small characters 236–39
resetting 54
resetting, with programs 240
setup menu 241–58, 338–40
print queue/spool. *See* Print command
PRN device name 14, 43–44
PRNTMENU.BAT file 247–52, *438–39*
PRNTREST.COM program 240–41
problems, computer
anticipating 354–57
formatting and unformatting disks 365–69
managing 357–59
preventing 359–65
printer compatibility 51–53
undeleting files 369–71
programs. *See also* files; software
accepting batch file interactions with 112–18, 127–32
creating, with Debug 74–80
cursor control with 211–15
filling screens of characters with 74–79
increasing keyboard repeat action with 282–83
programs, *continued*
memory-resident (*see* memory-resident programs)
memory usage by 168–70
overview *ix–x*
resetting IBM graphics printer with 240–41
restarting DOS with 190–91
search paths for 89, 144–45, 147–48
subdirectories for 291
turning off Num Lock with 283–85
Prompt command
command characters *25, 207*
customizing system prompts 205–10
redefining keyboards 276–81
sending ANSI commands with 23–25, 209–10
PROMPT environment variable 145, 146–47
PROMPTRS.BAT file 25, 146–47
prompts, system
colors in 209–10
customizing 205–10
default 147
defining and restoring 24–25
environment variables for 145, 146–47
multiple command processors and 99–100
standard colors and 220
protected mode 163
PS/2 keyboards 281–83

Q

q (Debug Quit command) 69
QRAM software 183–84
Qualitas 183
Quarterdeck Office Systems 183–84
queues, print. *See* Print command
quick-formatting disks 365–66
Quit command (Debug) 69

R

r (Debug Register command) 71–74
RAM (random access memory) 151, 161–62
RAM disks
advantages and disadvantages 106, 186, 391
creating 187–88
creating, with batch files 189–99

RAM disks, *continued*
DOS versions of 151
menu systems and 346–50
Mirror and 362
using 188–89
Windows and 373–74
RAMDRIVE.BAT file 189–99
RAMDRIVE.SYS device driver, 151, 187–88, 373–74
read-only file/directory attributes 87, 291
real mode (Windows) 373
rebooting DOS. *See* booting DOS
Recover command 359
recovery utilities 358–59
redirection. *See* input/output redirection
redirection operators (<, >, >>) 43, 77, 104
reformatting disks 365–66
Register command (Debug) 71–74
registers, changing contents of 71–74
repeat action, keyboard 281–83
replaceable parameters (%) 91, 105
Replace command 358
REPLY.COM program
accepting any key 115–16
accepting certain keys 116–18
accepting single keys 118
creating, with Debug 113–15
errorlevels and 112–13
rerouting. *See* input/output redirection
reserved memory. *See* upper memory area; upper memory blocks (umbs)
Reset command (printers) 54
RESET.COM program 190–91
restarting DOS. *See* booting DOS
Restore command 317, 327, 359
restoring files/directories 317, 327, 359. *See also* backing up files/directories
reverse video 27
ROM (read-only memory) 161–62
root directory 290
Running MS-DOS ix, 131, 350

S

screens
attributes (*see* display attributes)
blanking 215–17, 391
screens, *continued*
colors (*see* colors)
controlling cursors (*see* cursor control)
control program (*see* ANSI.SYS device driver)
designing, for batch files 85–86
displaying files on 34–35, 88–89
setting number of lines 204–5
suppressing messages 85, 88, 90, 91–92
system prompts (*see* prompts, system)
script files, Debug 74–76
search paths 89, 144–45, 147–48
secondary caches 151, 152
security systems 125–26, 132–34, 136–38, 216–17
segments, memory 62
Set command 144–45
SETVER.EXE device driver and command 151, 385–87
Share command 153
Shell configuration command 154–55
shell programs, 154. *See also* COMMAND.COM files; DOSShell command
SHOWMENU.BAT file 17–18
SHOWPATH.BAT file 147–48
SMARTDRIVE.SYS device driver 151, 177, 200–201, 373–74
software
boot monitor and config juggler programs 384
colors and 221
device drivers 183–84
extended DOS environments 78
hard-drive backup 317
Sound command 93–94
sound-generation 92–94
specifications
expanded memory (EMS) 162–63
extended memory (XMS) 156, 164
spooler, print. *See* Print command
STAMP.BAT file 307–8
stamping files with date and time 103–6, 307–8
standard ASCII characters 7, *424–25*
standard colors 219–20

standard mode (Windows) 372–73
starting DOS. *See* booting DOS
startup disk strategy 177–80
static electricity 392–93
strategies
  backup 316–17, 323–24, 326–27
  configuration 177–80
  memory expansion 182–84, 390–91
  memory optimization (*see* memory, optimizing)
subdirectories. *See* directories
Substitute command (subst) 362
surge suppressors 394
swap files, Windows 374–75
switches, protecting computer 391, 395–96
Syntax error message 88, 90
SYS extension 20
system clock. *See* date; time
system prompts. *See* prompts, system

T

Tab key 6, 14
TEMP environment variable 145, 375
templates, DOS command 260, 261
temporary files 145
terminate-and-stay-resident (TSR) programs. *See* memory-resident programs
text characters 7, *424–25*
text files. *See also* batch files
  adding date and time to 307–8
  adding text to beginning or end of 298–301
  ANSI.SYS commands in 24, 31–35
  carriage return as file contents 103
  combining 104, 314–16
  creating, with batch files 295–98
  creating, with console 59
  creating fixed-format memo 303–7
  creating note 301–3
  creating zero-length 137–38
  Debug script 74–76
  displaying contents of 88–89, 92
  printer commands in 46
  printing, in pages 230–36
third-party software. *See* software
time
  stamping files with 103–6, 307–8
  in system prompts 205–9
Time command 103
timing commands 105–6
TSR programs. *See* memory-resident programs
Type command
  creating zero-length files 137–38
  displaying contents of files 59, 88–89
  displaying contents of multiple files, 92
  Echo command vs. 247
  sending ANSI commands to consoles 34–35
typeface commands, printer
  Epson-compatible *411*
  HP-compatible *413*
  overview 42
  on printer setup menu *242*, 253–54

U

u (Debug Unassemble command) 80
UNBOX.BAT file 278–81
unconditional formatting 366
Undelete command 360, 363–65, 369–71
Unformat command 360, 366–69
uninterruptible power supplies 394–95
upper memory area 162
upper memory blocks (umbs)
  creating 174–75
  examining 166–70
  moving device drivers into 176–77
  moving Doskey into 265
  moving keyboard support into 288
  moving memory-resident programs into 158, 181–82
  preparing DOS to create 156–57, 172–73
utilities, recovery 358–59

V

variables, environment. *See* environment variables
VDISK.SYS device driver 151, 187
versions, DOS
  differences 356
  enabling 151, 385–87

VGA displays 204–5
video memory 184
virtual disks. *See* RAM disks

W

w (Debug Write command) 68–69
WINA20.386 file 372
Windows
    disk caching 200–201
    DOS 5 and 376–78
    EGA displays and 376
    environment variables 145
    extended vs. expanded memory 158, 174–75
    high memory area usage 156, 163
    memory usage 375–76
    modes of operation 372–73
    optimizing performance of 373–75
word processors
    entering ANSI.SYS commands 24, 31–32
    searching for control characters 15
Write command (Debug) 68–69

X

Xcopy command 318, 327
XMS (extended memory specification) 156, 164

Z

zero-length files 137–38

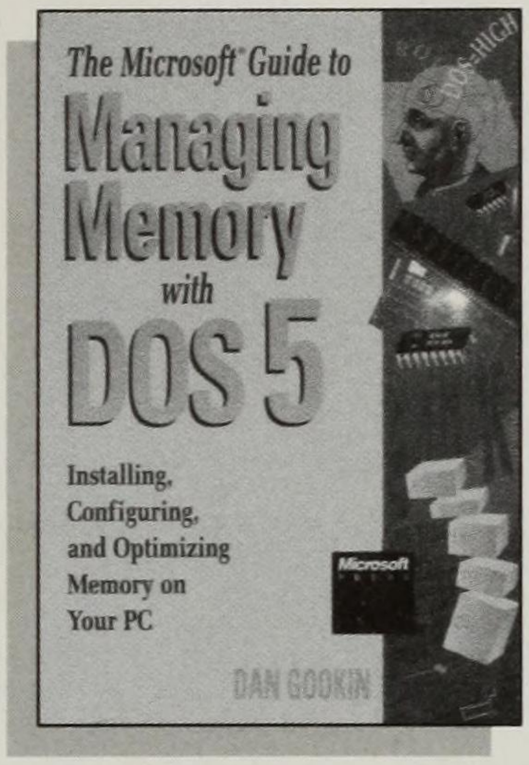
VAN WOLVERTON

A professional writer since 1963, Van Wolverton has had bylines as a newspaper reporter, editorial writer, political columnist, and technical writer. He wrote his first computer program—one that tabulated political polls—for the *Idaho State Journal* in Pocatello, Idaho, in 1965. His interests in computers and writing have been intertwined ever since. As a computer professional, Wolverton has worked at IBM and Intel and has written software documentation for the major national software companies, including Microsoft Corporation. He is also the author of the bestselling *Running MS-DOS* and was a contributor to *The MS-DOS Encyclopedia.* Wolverton and his wife, Jeanne, live in a twenty-first–century log cabin near Alberton, Montana.

DAN GOOKIN

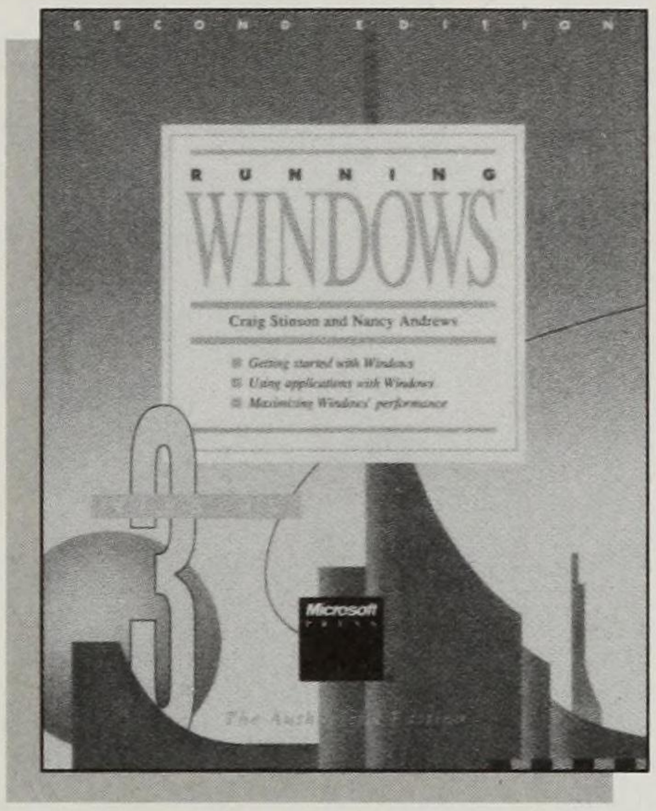
Dan Gookin is a free-lance writer specializing in books and articles about MS-DOS and IBM Personal Computers. In addition to his writing duties and passion for Preston Sturges films, he hosts a Sunday radio talk show in the San Diego area about personal computers. Gookin is the author of *The Microsoft Guide to Managing Memory with DOS 5,* published by Microsoft Press.

Great Resources from Microsoft Press!



Packed with Advice!

The Microsoft® Guide to
MANAGING MEMORY WITH DOS 5
Dan Gookin
208 pages, softcover 6 x 9 $14.95
Order Code GUMAME



Microsoft-Authorized Edition!

RUNNING WINDOWS,™ 2nd ed.
Craig Stinson and Nancy Andrews
544 pages, softcover $24.95
Order Code RUWI2

(See back for more details.)

Available wherever computer books are sold. Or order directly from Microsoft Press.

To order, call 1-800-MSPRESS or mail this order form:*

Quantity	Order Code	Title	Price	Total Price
_____	GUMAME	The Microsoft Guide to Managing Memory	$14.95	$ _____
_____	RUWI2	Running Windows, 2nd ed.	$24.95	$ _____

SALES TAX CHART	SHIPPING
Add the applicable sales tax for the following states: AZ, CA, CO, CT, DC, FL, GA, HI, ID, IL, IN, KS, KY, ME, MD, MA, MI, MN, MO, NE, NV, NJ, NM, NY, NC, OH, OK, PA, RI, SC, TN, TX, VA, WA, WV, and WI.	One book ..... $2.50 Two books ..... $3.25 Each additional book ..... $.75

SUBTOTAL	$	_____
Sales Tax	$	_____
Shipping	$	_____
TOTAL	$	_____

BBK

NAME _____

COMPANY (if applicable) _____

STREET (No P.O. Boxes) _____

CITY _____ STATE _____ ZIP _____

DAYTIME PHONE _____

CREDIT CARD NUMBER _____ EXP. DATE _____

CARDHOLDER SIGNATURE _____

PAYMENT:

- ☐ Check/Money Order (U.S. funds)
- ☐ VISA (13 or 16 digits)
- ☐ MasterCard (16 digits)
- ☐ American Express (15 digits)

FOR FASTER SERVICE CALL
1-800-MSPRESS
(8AM to 5 PM Central Time)
and place your credit card order.

Refer to campaign BBK.

All orders shipped RPS or UPS.
No P.O. Boxes please.
Allow 2–3 weeks for delivery.

*In Canada, contact Macmillan of Canada,
Attn: Microsoft Press Dept., 164 Commander Blvd.,
Agincourt, Ontario, Canada M1S 3C7 416-293-8141

In the U.K., contact Microsoft Press,
27 Wrights Lane, London W8 5TZ

The Microsoft® Guide to MANAGING MEMORY WITH DOS 5

Dan Gookin

"Now you can get your full dollar's worth out of all that memory in your computer."
From the Introduction

One of the most significant features of DOS 5 is its ability to use extended and expanded memory to effectively shatter the 640K barrier. If you're a beginning to intermediate DOS 5 user this official guide provides clear information on how this is done. Here's what's covered:

- the basics of memory—what is it and how your computer uses it
- the differences between conventional, extended, and expanded memory—in clear English
- buying, installing, and using RAM chips and memory boards
- tips on running the Windows environment more efficiently with DOS 5
- and much more

A great little book packed with advice.

RUNNING WINDOWS,™ 2nd ed.

Craig Stinson and Nancy Andrews

"This book presents the most thorough coverage of working with Microsoft Windows 3.0....written in a clear and concise manner...This book opens up Windows for all levels of users." **Bay Area Computer Currents**

Build your confidence and enhance your productivity with the Microsoft Windows environment—quickly and easily—using this hands-on introduction. This Microsoft-authorized edition is completely updated and expanded to cover all the exciting features of version 3. You'll find a successful combination of step-by-step tutorials, helpful screen illustrations, and real-world examples that will help you

- expertly maneuver through the Windows environment
- quickly master each Windows built-in desktop accessory
- tailor the Windows environment to suit your work habits
- use Windows on a network
- master the rich features of Windows' built-in word processing, communications, and graphics applications
- and much, much more

A special section includes techniques and strategies to help you optimize the performance of the Windows environment with extended and expanded memory, swap files, Smartdrive, and RAMdrive. If you're a novice Windows user, this book is the perfect starting point for exploring and understanding Windows. If you're a more experienced Windows user or familiar with an earlier version, you'll appreciate the in-depth advice, expert tips, and inside information.

ORDER CARD

For information on the Companion Disk for SUPERCHARGING MS-DOS, 3RD ED. and ordering information outside of the U.S., please see page xiv.

YES... please send me _____ copies of the Companion Disk for supercharging MS-DOS, 3rd ed. at $19.95 each (U.S. funds only) . . . . . $ _____

Sales Tax: Add the applicable sales tax in the following states: AZ, CA, CO, CT, DC, FL, GA, HI, ID, IL, IN, IA, KS, KY, ME, MD, MA, MI, MN, MO, NE, NV, NJ, NM, NY, NC, OH, OK, PA, RI, SC, TN, TX, VA, WA, WV, and WI. . . . . $ _____

Postage and Handling Charges: $2.50 per disk set (domestic orders)
$15.00 per disk set (foreign orders) . . . . . $ _____

Please check correct box: ☐ 5.25-inch format 097-000-573 ☐ 3.5-inch format 097-000-576 **TOTAL** $ _____

NAME

ADDRESS

( )
DAYTIME PHONE NUMBER

CITY STATE ZIP

Payment: ☐ Check/Money Order ☐ VISA (13 or 16 numbers) ☐ MasterCard (16 numbers) ☐ American Express (15 numbers)

Credit Card No. [1][2][3][4][5][6][7][8][9][10][11][12][13][14][15][16] _____ EXP. DATE

CARDHOLDER SIGNATURE

Please allow 2-3 weeks for delivery.

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST-CLASS MAIL PERMIT NO. 53 BOTHELL, WA

POSTAGE WILL BE PAID BY ADDRESSEE

MICROSOFT PRESS
ATTN: SUPERCHARGING MS-DOS,
3RD EDITION
COMPANION DISK OFFER
PO BOX 3011
BOTHELL WA 98041-9908

Craig Stinson and Nancy Andrews

"This book presents the most thorough coverage of working with Microsoft Windows 3.0....written in a clear and concise manner...This book opens up Windows for all levels of users." **Bay Area Computer Currents**

Build your confidence and enhance your productivity with the Microsoft Windows environment—quickly and easily—using this hands-on introduction. This Microsoft-authorized edition is completely updated and expanded to cover all the exciting features of version 3. You'll find a successful combination of step-by-step tutorials, helpful screen illustrations, and real-world examples that will help you

- expertly maneuver through the Windows environment
- quickly master each Windows built-in desktop accessory
- tailor the Windows environment to suit your work habits
- use Windows on a network
- master the rich features of Windows' built-in word processing, communications, and graphics applications
- and much, much more

A special section includes techniques and strategies to help you optimize the performance of the Windows environment with extended and expanded memory, swap files, Smartdrive, and RAMdrive. If you're a novice Windows user, this book is the perfect starting point for exploring and understanding Windows. If you're a more experienced Windows user or familiar with an earlier version, you'll appreciate the in-depth advice, expert tips, and inside information.

Microsoft®

THIRD EDITION

SUPERCHARGING MS-DOS®

COVERS VERSION 5

If you're a seasoned computer user and you're ready for *more*, this is the book for you!

This guide to the more powerful features of DOS* picks up where the classic *Running MS-DOS* leaves off. Here you'll find accurate, clear explanations of the advanced capabilities of the MS-DOS operating system—information you need to get the maximum in efficiency, productivity, and creativity from your PC.

In this new edition, the authors address the exciting improvements in MS-DOS 5 as well as recent developments in video, memory, and printer technology. Here's where to look for in-depth information on some of the most significant new developments in personal computing, supported by scores of new batch files and examples. Topics include:

- optimizing and customizing your system with the new features of DOS 5
- taking control of your screen display and keyboard using ANSI.SYS
- controlling Epson®–and Hewlett-Packard®–compatible printers and high-resolution color monitors through DOS commands and batch files
- using the DOSKEY utility to create and run keyboard macros
- breaking the 640-KB barrier with expanded and extended memory
- using Debug scripts and advanced batch-file techniques to create a menu system
- configuring DOS to make the most of the Microsoft® Windows™ environment

Move up to the power-user class with SUPERCHARGING MS-DOS.

*DOS as used here refers to the MS-DOS and PC-DOS operating systems.

Includes information on ordering the optional companion disk with ready-to-run utilities.

U.S.A.	**$24.95**
U.K.	£19.95
Canada	$32.95

[*Recommended*]

Microsoft PRESS®

The Authorized Editions

ISBN 1-55615-371-6

90000

9 781556 153716

Operating Systems/MS-DOS